Python basic to advance Bangla note

# স্ট্রিং নিয়ে কাজকারবার

Published by [subeen](http://pybook.subeen.com/author/subeen/) on [April 15, 2018](http://pybook.subeen.com/2018/04/)

স্ট্রিং (string)-এর সঙ্গে আমরা ইতিমধ্যেই পরিচিত হয়েছি। এই অধ্যায়ে আমরা স্ট্রিং-এর আরো ব্যবহার শিখবো। বেশিরভাগ ক্ষেত্রেই আমরা আলাদা ফাইলে প্রোগ্রাম না লিখে ইন্টারপ্রেটার ব্যবহার করবো, যাতে আমরা উদাহরণগুলো নিজেরা দ্রুত অনুসরণ করতে পারি।

শূণ্য বা তারচেয়ে বেশি সংখ্যক অক্ষর বা চিহ্ন দিয়ে স্ট্রিং তৈরি হয়। স্ট্রিং শুরু ও শেষ করতে হয় ডবল কোটেশন অথবা সিঙ্গেল কোটেশন চিহ্ন ব্যবহার করে। স্ট্রিংয়ে মোট কয়টি অক্ষর আছে, সেটি আমরা বের করতে পারি len() ফাংশন ব্যবহার করে।

>>> s = "hello"

>>> len(s)

5

>>> l = len(s)

>>> l

5

>>> print(l)

5

>>>

>>> s = ''

>>> len(s)

0

>>>

>>> s = ""

>>> len(s)

0

>>>

আবার আমাদের যদি এমন কিছু প্রিন্ট করতে হয়, যেখানে সিঙ্গেল কোটেশন আছে (যেমন Dimik’s), তখন আমরা সেটি দুইভাবে করতে পারি :

>>> s = "Dimik's"

>>> print(s)

Dimik's

>>> s = 'Dimik\'s'

>>> print(s)

Dimik's

>>>

একটি স্ট্রিংকে যদিও আমরা একটি ভ্যারিয়েবলে রাখি, আমরা চাইলে প্রতিটি অক্ষর আলাদাভাবে পেতে পারি। s যদি একটি স্ট্রিং হয়, তাহলে প্রথম অক্ষর পাওয়া যাবে s[0]-তে, দ্বিতীয় অক্ষর পাওয়া যাবে s[1]-এ।

>>> country = "Bangladesh"

>>> country[0]

'B'

>>> country[1]

'a'

>>> country[2]

'n'

>>> country[6]

'd'

>>> country[8]

's'

>>> country[9]

'h'

>>> country[10]

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

IndexError: string index out of range

ওপরের উদাহরণ থেকে আমরা বুঝতে পারছি, স্ট্রিংয়ের দৈর্ঘ্য যদি len হয়, তাহলে প্রথম অক্ষরটি থাকবে 0 ইনডেক্সে আর শেষ অক্ষরটি থাকবে len-1 ইনডেক্সে। আমরা স্ট্রিংয়ের ওপর লুপও চালাতে পারি এভাবে :

>>> for c in country:

... print(c)

...

B

a

n

g

l

a

d

e

s

h

>>>

অনেকটা লিস্টের মতোই। কিন্তু লিস্টের সঙ্গে একটি পার্থক্য হচ্ছে, আমরা চাইলে লিস্টের কোনো উপাদান পরিবর্তন করতে পারি, স্ট্রিংয়ের ক্ষেত্রে সেটি সম্ভব নয়।

>>> c = ['A', 'b', 'c']

>>> print(c)

['A', 'b', 'c']

>>> c[0] = 'a'

>>> print(c)

['a', 'b', 'c']

>>>

>>> country = "Bangladesh"

>>> country[0] = 'b'

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'str' object does not support item assignment

>>>

এজন্য পাইথনের ইংরেজি বইগুলোতে স্ট্রিংকে বলে ইমমিউটেবল (immutable) বা নন-মিউটেবল (non-mutable), কারণ এর পরিবর্তন করা যায় না।

দুটি স্ট্রিং জোড়া লাগানোর জন্য আমরা যোগ চিহ্ন ব্যবহার করতে পারি।

>>> country = "Bangla" + "desh"

>>> print(country)

Bangladesh

>>> x = "50" + "5"

>>> print(x)

505

>>>

এই যোগের কাজটা করতে গিয়ে আমরা অনেকসময় গড়বড় করে ফেলি। তাই একটু সতর্কতা অবলম্বন করতে হবে যে, আমরা আসলে সংখ্যা যোগ করছি নাকি স্ট্রিং জোড়া লাগাচ্ছি।

অনেকসময় আমাদেরকে একটি স্ট্রিংয়ের ভেতরে বিশেষ একটি স্ট্রিং খুঁজে বের করতে হবে। তখন আমরা find() মেথড ব্যবহার করবো। যেমন :

>>> country = "Bangladesh"

>>> country.find("Ban")

0

>>> country.find("ang")

1

>>> country.find("Bangla")

0

>>> country.find("Bengla")

-1

>>> country.find("desh")

6

এখানে আমরা country (অর্থাৎ Bangladesh)-এর ভেতরে বিভিন্ন স্ট্রিং খুঁজে বের করার চেষ্টা করেছি। যদি আমরা যা খুঁজছি, সেটি পাওয়া না যায়, তখন find() মেথড -1 রিটার্ন করে। আর যদি খুঁজে পাওয়া যায়, তাহলে যেই পজিশনে সেটি খুঁজে পাওয়া গিয়েছে, সেই পজিশন বা ইনডেক্স রিটার্ন করে।

আমরা যদি কোনো একটি স্ট্রিংয়ের ভেতরে কোনো কিছু বদলে দিতে চাই, তাহলে আমরা replace() মেথড ব্যবহার করে সেটি করতে পারি। যেমন :

>>> country = "North Korea"

>>> new\_country = country.replace("North", "South")

>>> print(new\_country)

South Korea

>>> print(country)

North Korea

>>>

>>> text = "this is a test. this is another test. this is final test."

>>> new\_text = text.replace("this", "This")

>>> print(new\_text)

This is a test. This is another test. This is final test.

>>> print(text)

this is a test. this is another test. this is final test.

ওপরের উদাহরণে আমরা দেখলাম যে, replace() মেথডটি নতুন একটি স্ট্রিং রিটার্ন করে, কিন্তু মূল স্ট্রিংয়ে কোনো পরিবর্তন করে না। তবে আমরা চাইলে আগের স্ট্রিং যেই ভ্যারিয়েবলে রেখেছিলাম, নতুন স্ট্রিংটিও একই ভ্যারিয়েবলে রাখতে পারি। সেক্ষেত্রে একই নামের একটি নতুন ভ্যারিয়েবলে নতুন স্ট্রিং অ্যাসাইন হবে। যদিও আমাদের মনে হতে পারে আগের স্ট্রিং পরিবর্তিত হয়েছে, সেটি আসলে সত্যি নয়, কারণ স্ট্রিং পরিবর্তন করা যায় না, একটু আগেই আমরা সেটি দেখেছি।

>>> text = "hello"

>>> text = text.replace("hello", "Hello")

>>> print(text)

Hello

ওপরে ঘটনা যা ঘটছে, তা হলো, text.replace(“hello”, “Hello”) একটি নতুন স্ট্রিং তৈরি করছে, আর সেটিকে আমরা text ভ্যারিয়েবলেই রাখছি। “hello” স্ট্রিংটি বদলায়নি, বরং নতুন একটি স্ট্রিং তৈরি হয়েছে – এই বিষয়টি খেয়াল রাখতে হবে। প্রোগ্রামিংয়ে যারা নতুন, তাদের হয়ত বুঝতে একটু সমস্যা হতেও পারে, তাতে কোনো অসুবিধা নেই।

স্ট্রিংয়ের শুরুতে ও শেষে অনেক সময় স্পেস ক্যারেক্টার থাকে, যেগুলো আমাদের বাদ দেওয়ার প্রয়োজন হতে পারে। সেজন্য আমরা strip(), lstrip() ও rstrip() মেথড ব্যবহার করতে পারি। lstrip() মেথড স্ট্রিংয়ের বাঁ দিকের স্পেসগুলো বাদ দেবে, rstrip() মেথড ডানদিকের স্পেসগুলো বাদ দেবে, আর strip() মেথড দুইদিকের স্পেসগুলোই বাদ দেবে।

>>> text = " this is a string. "

>>> text

' this is a string. '

>>> text.lstrip()

'this is a string. '

>>> text.rstrip()

' this is a string.'

>>> text.strip()

'this is a string.'

>>> text

' this is a string. '

>>>

ওপরের উদাহরণে আমরা আরো দেখলাম যে text-এর কিন্তু কোনো পরিবর্তন হলো না। কারণ মেথডগুলো আসলে নতুন একটি স্ট্রিং রিটার্ন করে। তাই সেগুলো পেতে চাইলে নতুন কোনো ভ্যারিয়েবলে (কিংবা text ভ্যারিয়েবলে) রাখতে হবে।

>>> text = " this is a string. "

>>> new\_text = text.rstrip()

>>> new\_text

' this is a string.'

>>> text

' this is a string. '

>>>

আবার আমরা যদি কোনো স্ট্রিংকে আপারকেস (uppercase), লোয়ারকেস (lowercase)-এ রূপান্তর করতে চাই, কিংবা কেবল প্রথম অক্ষরটি বড় হাতের করে দিতে চাই, সেজন্যও ফাংশন তৈরি করা আছে।

>>> s1 = "Bangladesh"

>>> s\_up = s1.upper()

>>> s\_up

'BANGLADESH'

>>> s\_lo = s1.lower()

>>> s\_lo

'bangladesh'

>>> s = "hello"

>>> s\_cap = s.capitalize()

>>> s\_cap

'Hello'

কোনো স্ট্রিংয়ে যদি অনেকগুলো ছোট স্ট্রিং এক বা একাধিক স্পেস দিয়ে আলাদা করা থাকে আর আমরা কেবল সেই ছোট স্ট্রিংগুলো পেতে চাই, তাহলে আমরা split() মেথড ব্যবহার করে সেগুল একটি লিস্টে পেতে পারি।

>>> str = "I am a programmer."

>>> words = str.split()

>>> print(words)

['I', 'am', 'a', 'programmer.']

>>> words

['I', 'am', 'a', 'programmer.']

>>> for word in words:

... print(word)

...

I

am

a

programmer.

>>>

স্ট্রিংয়ের ভেতরের স্ট্রিংকে সাবস্ট্রিং (sub-string)ও বলে। সেটের যেমন সাবসেট। এখন কোনো স্ট্রিংয়ের ভেতরে একটি স্ট্রিং কতবার আছে, সেটি গণনা করারও একটি সহজ উপায় আছে, count() মেথডের মাধ্যমে।

>>> str = "This is"

>>> str.count("is")

2

কখনো কখনো আমাদের জানার দরকার হতে পারে যে, একটি স্ট্রিংয়ের শুরুতে কিংবা শেষে একটি স্ট্রিং (বা সাবস্ট্রিং) আছে কী নেই। সেজন্য পাইথনে দুটি মেথড তৈরি করা আছে, startswith() ও endswith()। নিচের উদাহরণ দেখলেই বুঝতে পারবো।

>>> s = "Bangladesh"

>>> s.startswith("Ban")

True

>>> s.startswith("ban")

False

>>> s.startswith("an")

False

>>> s.endswith("Ban")

False

>>> s.endswith("desh")

True

>>> s.endswith("h")

True

আমাদের হয়ত এমন লজিকের দরকার হতে পারে যে কারো নাম Mr. দিয়ে শুরু হলে আমরা তাকে Dear Sir বলে সম্বোধন করবো। তখন এরকম কোড লেখা যেতে পারে :

>>> name = "Mr. Anderson"

>>> if name.startswith("Mr."):

... print("Dear Sir")

...

Dear Sir

ওপরের প্রোগ্রামটিতে ইনপুট নেওয়ার জন্য আমরা টেক্সটবক্সও ব্যবহার করতে পারি। নিচে টার্টল মডিউল ব্যবহার করে আমরা টেক্সট ইনপুট নিলাম। একটি প্রোগ্রাম অনেকভাবেই লেখা যায়, এটি তার একটি ভালো উদাহরণ।

import turtle

name = turtle.textinput("name", "What is your name?")

name = name.lower()

if name.startswith("mr"):

print("Hello Sir, how are you?")

elif name.startswith("mrs") or name.startswith("miss") or name.startswith("ms"):

print("Hello Madam, how are you?")

else:

name = name.capitalize()

str = "Hi " + name + "! How are you?"

print(str)

turtle.exitonclick()

এতক্ষণ স্ট্রিংয়ের নানান রকমের কাজ দেখলাম। ভবিষ্যতে বিভিন্ন প্রোগ্রাম তৈরি করার সময় এগুলো কাজে লাগবে। আর বইতে যা দেখানো হয়েছে, তার বাইরেও অনেক কিছু আছে, সেগুলো আগামী দিনগুলোতে শিখে নিতে হবে। প্রোগ্রামিং নিয়ে তাড়াহুড়া করার কিছু নেই। প্রচুর সময়, ধৈর্য্য, চর্চা ও লেখাপড়া করার বিনিময়ে ভালো প্রোগ্রামার হওয়া সম্ভব। এখন একটি মজার প্রোগ্রাম দিয়ে এই অধ্যায়ের ইতি টানবো।

str = "a quick brown fox jumps over the lazy dog"

for c in "abcdefghijklmnopqrstuvwxyz":

print(c, str.count(c))

প্রোগ্রামটি রান করে আউটপুট দেখে বুঝতে হবে কেন এটি একটি মজার প্রোগ্রাম।

**অনুশীলনী:**

* একটি প্রোগ্রাম লিখতে হবে, যা ইনপুট হিসেবে একটি স্ট্রিং নেবে এবং সেই স্ট্রিং থেকে মোট চারটি স্ট্রিং তৈরি করে স্ক্রিনে প্রিন্ট করবে। প্রথম স্ট্রিংটি হবে কেবল বড় হাতের অক্ষরগুলো নিয়ে, দ্বিতীয় স্ট্রিং হবে কেবল ছোট হাতের অক্ষরগুলো নিয়ে, তৃতীয় স্ট্রিং হবে কেবল অংক (digit) নিয়ে আর বাকী সবকিছু চতুর্থ স্ট্রিংয়ে থাকবে। যেমন, ইনপুট যদি Hello Test! 123 123, good. হয়, তাহলে আউটপুট হবে এমন –
* HT
* elloestgood
* 123123
* ! , .
* একটি প্রোগ্রাম লিখতে হবে, যা ইনপুট হিসেবে একটি স্ট্রিং নেবে এবং প্রথম ও দ্বিতীয়, তৃতীয় ও চতুর্থ, পঞ্চম ও ষষ্ঠ – এভাবে পাশাপাশি দুটি অক্ষরকে অদল-বদল করে একটি নতুন স্ট্রিং তৈরি করবে। যেমন, ইনপুট যদি হয় Bangladesh, তাহলে আউটপুট হবে aBgnaledhs।
* কোনো শব্দকে উল্টো করে সাজালেও যদি আগের মতোই শব্দ পাওয়া যায়, তাকে বলে প্যালিনড্রোম ()। যেমন: madam শব্দটির অক্ষরগুলো উল্টো করে সাজালে আমরা madam পাই। তাই madam একটি প্যালিনড্রোম। আবার toyota শব্দটির অক্ষরগুলো উল্টো করে সাজালে পাওয়া যায় atoyot, তাই এটি প্যালিন্ড্রোম নয়। এখন, একটি প্রোগ্রাম লিখতে হবে, যেটি ব্যবহারকারীর কাছ থেকে একটি স্ট্রিং ইনপুট নেবে এবং স্ট্রিংটি প্যালিনড্রোম কী না, সেটি বলে দেবে।

# ক্লাস, অবজেক্ট এবং মেথড

class WaltonUsta:

"""Our new car class"""

def driving(self):

run\_the\_car

def music(self):

run\_the\_music

def fuel(self):

load\_the\_fuel

def horn(self):

make\_sound\_pollution

আমরা এখানে WaltonUsta নামে একটা ক্লাস তৈরি করেছি আর তার ভিতর driving(), music(), fuel(), horn() চারটা ফাংশন তৈরি করেছি। এই নিয়ে হয়ে গেল আমাদের ক্লাস।

### সেলফ (self)

আমি নিশ্চিত আমাদের অনেকেরই self নিয়ে ব্যাপক দুশ্চিন্তা হচ্ছে। এটা নিয়ে মাথা ঘামানোর আসলে তেমন কিছু নাই। কারণ:

* self আসলে একটা কনভেনশনাল নাম। আমরা চাইলে এটাকে বদলে দিতে পারি।
* self হল ক্লাস ইন্সট্যান্সের একটা রেফারেন্স। ইন্সট্যান্স ভ্যারিয়েবল ডিক্লেয়ার করার সময় self এর রেফারেন্স ব্যবহার করা হয়। ক্লাস ভ্যারিয়েবল আর ইন্সট্যান্স ভ্যারিয়েবলের ভিতর পার্থক্য হল ক্লাস ভ্যারিয়েবল ক্লাসের সব ইন্সট্যান্সের ভিতরেই শেয়ার্ড, অন্যদিকে ইন্সট্যান্স ভ্যারিয়েবল প্রতিটা ইন্সট্যান্সে ইউনিক।

### অবজেক্ট (object) কী?

অবজেক্টের শাব্দিক অর্থ বস্তু। নামের মধ্যেই এর কাজের কথা বলা আছে। অবজেক্ট আসলে বস্তুবাদী মানে পার্থিব চিন্তার বিষয়। (পৃথিবীতে আমাদের সামনে দৃশ্যমান বস্তুই পার্থিব।)

আমাদের ঐ যে ‘ওয়ালটন উস্তা’ ক্লাসটা, এটা দিয়ে আমরা কিন্তু লাখ লাখ গাড়ি তৈরি করতে পারি। প্রতিটা গাড়িতেই কিন্তু ঐ চারটা ফাংশন থাকবে। এখন আমাদের গাড়িটার কথা ধরি। আমাদের গাড়িটাও তো ‘ওয়ালটন উস্তা’ ক্লাস থেকে তৈরি। তাহলে আমাদের গাড়িটা হল ঐ ক্লাসের একটা অবজেক্ট। ঐ ক্লাস থেকে তৈরি সব গাড়িই এক-একটা অবজেক্ট।

চলুন সবাই মিলে পাইথনিক উপায়ে ‘ওয়ালটন উস্তা'র অবজেক্ট তৈরি করি।

our\_car = WaltonUsta()

her\_car = WaltonUsta()

your\_car = WaltonUsta()

এখানে our\_car, her\_car, your\_car সবগুলো হল অবজেক্ট, WaltonUsta() ক্লাসের অবজেক্ট। আর প্রতিটা অবজেক্টই কিন্তু ক্লাসের সব ফাংশনকে অ্যাক্সেস করতে পারবে।

### মেথড (Method) কী?

মেথড নতুন কিছু না। এতক্ষণ আমরা যাকে ফাংশন নামে চিনেছি, সে যখন ক্লাসের ভিতর থাকে তখন তাকে আমরা মেথড বলব। শুধু নামকরণে তফাৎ। বাদবাকি সবকিছুই এক। অর্থাৎ, ক্লাসের ফাংশনকে মেথড বলে। উঁহু, আরেকটা তফাৎ আছে। সবসময় মেথডের প্রথম আর্গুমেন্ট বা প্যারামিটার হবে self। আর মেথড কল করার সময় এটা পাস করার দরকার নাই। পাইথন নিজ থেকেই এটা পাস করে দেবে।

একটা কথা হয়নি বলা। ক্লাসের নাম আমরা সবসময় বড় হাতের অক্ষর দিয়ে শুরু করব। অবজেক্টের নাম ভ্যারিয়েবলের মত করেই হবে আর মেথড তো ফাংশনের মত করেই। আপাতত এটুকু জানলেই চলবে আমাদের। নামকরণের নানা নিয়ম-কানুন আমরা কনভেনশন চাপ্টারে বিস্তারিত শিখব।

এখন আমরা বাস্তব একটা উদাহরণ দেখব। আমরা ক্যালকুলেটরের জন্য একটা ক্লাস তৈরি করব। এই ক্লাসে যোগ, বিয়োগ, গুণ ও ভাগের জন্য মেথড থাকবে।

class Calculator:

"""Do addition, subtraction, multiplication and division."""

def addition(self, a, b):

return a+b

def subtraction(self, a, b):

return a-b

def multiplication(self, a, b):

return a\*b

def division(self, a, b):

try:

return a/b

except ZeroDivisionError:

return 'It is impossible to divide by zero.'

my\_calculator = Calculator()

temp = my\_calculator.addition(12, 78)

print(temp)

temp = my\_calculator.subtraction(50, 23)

print(temp)

temp = my\_calculator.multiplication(9, 19)

print(temp)

temp = my\_calculator.division(400, 5)

print(temp)

temp = my\_calculator.division(43, 0)

print(temp)

**আউটপুট**

90

27

171

80.0

It is impossible to divide by zero.

Example

class car:

**def** band(*self*,**name**):

*self*.name**=**name

        print(name)

**def** color(*self*,**color**):

*self*.color**=**color

        print(color)

**def** price(*self*,**taka**):

*self*.taka**=**taka

        print(taka)

lol**=**car()

tol**=**lol.band('cool')

tol**=**lol.color('blue')

tol**=**lol.price(123456789)

এখানে আমরা কিভাবে আমাদের মেথডগুলো অ্যাক্সেস করেছি খেয়াল করেছি তো সবাই? অবজেক্টের পর . চিহ্ন দিয়ে মেথড অ্যাক্সেস করতে হয়। ইতিপূর্বে আমরা এই কাজ লিস্ট, টাপল, ডিকশনারি এসব জায়গায় করে এসেছি। আসলে এগুলো হল বিল্ট-ইন ক্লাস। পাইথনে সবকিছুই অবজেক্ট। আমরা তখন যাদের ফাংশন বলেছি আসলে তারা ছিল মেথড।

### \_\_init\_\_ constructor

যদি আমরা ক্লাসের ভিতর কোন ভ্যালু পাস করতে চাই? আর পরে যদি সেই ভ্যালু নিয়েই মেথডের ভিতর কোডিং কারিশমা ফলাতে চাই? তখন ক্লাসের প্রথম মেথড হিসাবে \_\_init\_\_ ব্যবহার করতে হবে। প্রোগ্রামিংয়ের ভাষায় একে ক্লাস কনস্ট্রাক্টর (constructor) বলে। যখন কোন ক্লাসের নতুন ইন্সট্যান্স বা অবজেক্ট তৈরি করি আমরা তখন পাইথন নিজ থেকেই ক্লাস কনস্ট্রাক্টরকে কল করে। ওহ! যখন ক্লাস কনট্রাক্টর আমরা ডিফাইন করি না তখন পাইথন নিজ থেকেই ক্লাস কনস্ট্রাক্টর ডিফাইন করে নেয়।

class Calculator:

"""Do addition, subtraction, multiplication and division."""

def \_\_init\_\_(self, a, b):

self.a = a

self.b = b

def addition(self):

return self.a + self.b

def subtraction(self):

return self.a - self.b

def multiplication(self):

return self.a \* self.b

def division(self):

try:

return self.a / self.b

except ZeroDivisionError:

return 'It is impossible to divide by zero.'

my\_calculator = Calculator(45, 3)

temp = my\_calculator.addition()

print(temp)

temp = my\_calculator.subtraction()

print(temp)

temp = my\_calculator.multiplication()

print(temp)

temp = my\_calculator.division()

print(temp)

**আউটপুট**

48

42

135

15.0

আমরা \_\_init\_\_ ব্লকটা রিভিউ করি একটু। বাধ্যতামূলক self বাদে আমরা আরো দুইটা প্যারামিটার a, b দিয়েছি। তারপর self.a তে a এর ভ্যালু ও self.b তে b এর ভ্যালু অ্যাসাইন করেছি। এর ফলে a, b এর ভ্যালু self.a, self.b এর মাধ্যমে পুরো ক্লাসের ভিতর অ্যাক্সেসিবল হয়ে গেল।

এবার ইনস্ট্যান্স বা অবজেক্ট ক্রিয়েশনটা রিভিউ করি। my\_calculator অবজেক্ট তৈরির সময় আমরা Calculator এর ভিতর দুইটা ভ্যালু পাস করেছি। এটার সাথে \_\_init\_\_ মেথডের যোগসাজশ আছে। এই ভ্যালু দুইটা সরাসরি \_\_init\_\_ এর ভিতর চলে আসে। সহজ কথায়, \_\_init\_\_ বা ক্লাস কনস্ট্রাক্টরের প্যারামিটারই ক্লাসের প্যারামিটার।

class Calculator:

    """Do addition, subtraction, multiplication and division."""

**def** \_\_init\_\_(*self*, **band**, **color**,**price**):

*self*.band **=** band

*self*.color **=** color

*self*.price**=**price

**def** addition(*self*):

        print(*self*.band )

**def** subtraction(*self*):

        print(*self*.color)

**def** multiplication(*self*):

        print(*self*.price)

my\_calculator **=** Calculator('bmw', 'red',3)

my\_calculator.addition()

my\_calculator.subtraction()

my\_calculator.multiplication()

## অবজেক্ট ও ক্লাস

প্রথমে বলে নিই, অবজেক্ট আর ইনস্ট্যান্স একই জিনিস। একেক সময় একেক নাম ব্যবহার করা হয়। আর যখনই অবজেক্ট (বা ইনস্ট্যান্স)-এর কথা বলা হয়, তার সঙ্গে আরেকটি জিনিস যুক্ত করে দিতে হয়, যে এই অবজেক্টটি কোন ক্লাসের অবজেক্ট? ক্লাস হচ্ছে মূল নকশা আর সেই ক্লাসের এক বা একাধিক অবজেক্ট তৈরি করা যায়, যারা ওই ক্লাসের সব বৈশিষ্ট্য ধারণ করবে। যেমন, একটি গাড়ির ডিজাইন হচ্ছে ক্লাস আর সেই ডিজাইন অনুসরণ করে যত গাড়ি তৈরি করা হয়, সেই গাড়িগুলো হচ্ছে ওই ক্লাসের অবজেক্ট। তেমনি turtle মডিউলের ভেতরে Turtle ক্লাসে বলা আছে যে, একটি টার্টলের কী কী বৈশিষ্ট্য থাকবে এবং সেটি কী কী কাজ করতে পারবে ও কিভাবে করতে পারবে। এই বৈশিষ্ট্যগুলোকে বলা হয় ডেটা অ্যাট্রিবিউট (data attribute) আর যেসব কাজ করতে পারবে, সেগুলোকে বলা হয় মেথড (method)। আসলে মেথডগুলো হচ্ছে ক্লাসের ভেতরে তৈরি করা ফাংশন, তবে ক্লাস ও অবজেক্টের বেলায় তাদেরকে আমরা মেথড বলি। একারণেই এই বইতে এবং আগের বইতে কোথাও কোথাও ফাংশন, কোথাও কোথাও মেথড শব্দ ব্যবহার করা হয়েছে।

>>> dir(li)

['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_',

'\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_',

'\_\_getitem\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_iadd\_\_', '\_\_imul\_\_', '\_\_init\_\_',

'\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_',

'\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_reversed\_\_', '\_\_rmul\_\_',

'\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_',

'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop',

'remove', 'reverse', 'sort']

যেসব অ্যাট্রিবিউটগুলোর নাম আন্ডারস্কোর চিহ্ন (\_) দিয়ে শুরু হয়েছে, সেগুলো আপাতত আমাদের বোঝার দরকার নেই। সেগুলো বাদ দিলে আমরা দেখতে পাচ্ছি, লিস্টের এই অ্যাট্রিবিউটগুলো : ‘append’, ‘clear’, ‘copy’, ‘count’, ‘extend’, ‘index’, ‘insert’, ‘pop’, ‘remove’, ‘reverse’, ‘sort’। এগুলোর বেশ কিছু ব্যবহার আমরা আগের বইতে দেখেছি। এখন চাইলে নিজে নিজে পরীক্ষানিরীক্ষা করে দেখা যেতে পারে যে, কোন অ্যাট্রিবিউট কী কাজ করে। আর পাইথনের অফিশিয়াল ডকুমেন্টেশনেও লিস্টের বিভিন্ন মেথডগুলোর কাজের বর্ণনা লেখা আছে। যেহেতু li হচ্ছে list ক্লাসের একটি অবজেক্ট, তাই এর বিভিন্ন মেথড ব্যবহার করার সময় আমরা প্রথমে লিখবো অবজেক্টের নাম, তারপরে ডট, তারপরে মেথডের নাম। যেমন : li.sort()।

>>> li.sort()

>>> li

[1, 2, 10, 20]

Example:

class Car:

name = "Premio"

color = "white"

def start():

print("Starting the engine")

ওপরে আমরা Car নামে একটি ক্লাস তৈরি করলাম। তারপরে সেখানে name ও color নামে দুটি ডেটা অ্যাট্রিবিউট রাখলাম, আর start নামে একটি মেথড তৈরি করলাম। এখন আমরা এই অ্যাট্রিবিউটগুলো ব্যবহার করতে পারবো

**\*\*\*\*Attribute and parameter are same jinis**

**\*\*\*\*অবজেক্ট তৈরির কাজটিকে পাইথনের ভাষায় বলে ইনস্ট্যানশিয়েট (instantiate) করা। এজন্য অবজেক্টকে ইনস্ট্যান্সও বলা হয়ে থাকে।**

class Car:

name = ""

color = ""

def \_\_init\_\_(self, name, color):

self.name = name

self.color = color

def start(self):

print("Starting the engine")

my\_car = Car("Corolla", "White")

print(my\_car.name)

print(my\_car.color)

my\_car.start()

প্রোগ্রামটি রান করলে ঠিকঠাক আউটপুট আসবে। এখানে আমি একটি নতুন মেথড ব্যবহার করেছি, \_\_init\_\_()। যখন কোনো ক্লাসের অবজেক্ট তৈরি করা হয়, তখন এই মেথডটি আপনাআপনি (অটোমেটিক) কল হয়। তাই যখন my\_car = Car(“Corolla”, “White”) স্টেটমেন্টটি চলে, তখন আসলে Car ক্লাসের \_\_init\_\_ মেথড কল হয়। সেই মেথডের প্রথম প্যারামিটার হচ্ছে self। এটি সবসময়ই দিতে হবে। তারপরে যেহেতু আমরা গাড়ির নাম ও রং সেট করে দিতে চাই, তাই name ও color নামে দুটি প্যারামিটার রাখছি। তারপরে ফাংশনের ভেতরে self.name-এ name অ্যাসাইন করছি, আর self.color-এ color অ্যাসাইন করছি। আমি যখন লিখছি self.name, সেটি বোঝাচ্ছে যে, যেই অবজেক্টটি আমি তৈরি করছি, তার name অ্যাট্রিবিউট। এখানে Car ক্লাসের ভেতরে যে name ডিক্লেয়ার করেছি (name = “”), self.name, আর init ফাংশনের name প্যারামিটার – তিনটি কিন্তু আলাদা। এখানে এসে ব্যাপারটা একটু এলোমেলো লাগতে পারে। তাই আমরা ওপরের কোডটি একটু পরিবর্তন করে লিখি।

class Car:

def \_\_init\_\_(self, n, c):

self.name = n

self.color = c

def start(self):

print("Starting the engine")

my\_car = Car("Corolla", "White")

print(my\_car.name)

print(my\_car.color)

my\_car.start()

প্রোগ্রামটি রান করলে আউটপুট আসবে এরকম –

$ python car.py

Corolla

White

Starting the engine

এখানে যখন আমি my\_car = Car(“Corolla”, “White”) লিখছি, তখন \_\_init\_\_(self, n, c)-এর self-এ যাচ্ছে my\_car অবজেক্টের রেফারেন্স, n-এ যাচ্ছে “Corolla”, আর c-তে যাচ্ছে “White”। তারপর যখন self.name = n স্টেটমেন্ট এক্সিকিউট হচ্ছে, তখন my\_car অবজেক্টের name নামে একটি অ্যাট্রিবিউট তৈরি হচ্ছে আর সেখানে n অ্যাসাইন হচ্ছে। একইভাবে color অ্যাট্রিবিউট তৈরি হয়ে সেখানে c-তে যা পাঠিয়েছিলাম, তা অ্যাসাইন হচ্ছে।

Car ক্লাসে যে দুটি ডেটা অ্যাট্রিবিউট তৈরি করেছিলাম (name ও color), সেগুলো কিন্তু আমি বাদ দিয়ে দিয়েছি। কারণ আমি অবজেক্ট তৈরি করার সময় init মেথডের ভেতরে self.name ও self.color যখন লিখছি, তখন সেই অবজেক্টের জন্য name ও color নামে অ্যাট্রিবিউট তৈরি হয়ে যাচ্ছে। একে বলে ইনস্ট্যান্স অ্যাট্রিবিউট, যা কেবল ওই ক্লাসের ইনস্ট্যান্সের (বা অবজেক্টের) থাকে। তবে এখন কিন্তু আর ক্লাসের নাম লিখে ডট দিয়ে name (ও color) একসেস করা যাবে না। আমরা যদি লিখি print(Car.name), তাহলে আমরা এরর পাবো : AttributeError: type object ‘Car’ has no attribute ‘name’। অর্থাৎ Car ক্লাসের name নামে কোনো অ্যাট্রিবিউট নেই। আমরা যখন এরকম লিখেছিলাম –

class Car:

name = ""

color = ""

তখন, এই name ও color–কে বলা হয় ক্লাস অ্যাট্রিবিউট। বেশিরভাগ সময়ই আমাদের এরকম অ্যাট্রিবিউট তৈরি করার দরকার পরবে না।

এখন ক্লাসের ভেতরে যেসব মেথড আছে, তারা যদি ওই ক্লাসের অবজেক্টের বিভিন্ন অ্যাট্রিবিউট একসেস করতে চায়, সেই কাজটি করতে পারবে, তবে অ্যাট্রিবিউটের নামের আগে অবশ্যই self লিখে ডট দিতে হবে, তাহলে পাইথন বুঝতে পারবে যে, অ্যাট্রিবিউটটি কোন অবজেক্টের সঙ্গে সংশ্লিষ্ট।

class Car:

def \_\_init\_\_(self, n, c):

self.name = n

self.color = c

def start(self):

print("name: ", self.name)

print("color: ", self.color)

print("Starting the engine")

my\_car = Car("Corolla", "White")

my\_car.start()

ওপরের প্রোগ্রামে আমরা start() মেথডের ভেতরে অবজেক্টের নাম ও রং প্রিন্ট করেছি। এখন একটি মজার জিনিস দেখাই। self-এ যে অবজেক্ট পাঠানো হয়, তার প্রমাণ পাওয়া যাবে নিচের প্রোগ্রাম চালালে –

class Car:

def \_\_init\_\_(self, n, c):

self.name = n

self.color = c

def start(self):

print("name: ", self.name)

print("color: ", self.color)

print("Starting the engine")

my\_car = Car("Corolla", "White")

Car.start(my\_car)

আমরা my\_car অবজেক্ট তৈরি করলাম। তারপর, Car.start() মেথড কল করলাম কিন্তু আর্গুমেন্ট হিসেবে my\_car অবজেক্ট ব্যবহার করলাম। এই প্রোগ্রামের আউটপুট কিন্তু হুবুহু আগের প্রোগ্রামের মতোই হবে। তবে বলে রাখা ভালো যে, সাধারণত এভাবে আমরা start() মেথড কল করবো না।

এখন আমরা চাইলে একাধিক কার অবজেক্ট তৈরি করতে পারি।

class Car:

def \_\_init\_\_(self, n, c):

self.name = n

self.color = c

def start(self):

print("name: ", self.name)

print("color: ", self.color)

print("Starting the engine")

my\_car1 = Car("Corolla", "White")

my\_car1.start()

my\_car2 = Car("Premio", "Black")

my\_car2.start()

my\_car3 = Car("Allion", "Blue")

my\_car3.start()

প্রোগ্রামটি রান করলে আউটপুট আসবে এরকম :

$ python car.py

name: Corolla

color: White

Starting the engine

name: Premio

color: Black

Starting the engine

name: Allion

color: Blue

Starting the engine

প্রতিটি অবজেক্টের অ্যাট্রিবিউটগুলো কিন্তু আলাদা। যেমন, my\_car1-এর color আর my\_car2-এর color আলাদা। এটি হচ্ছে অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিংয়ের একটি বড় সুবিধা। আমরা যত খুশি অবজেক্ট তৈরি করতে পারি। প্রতিটি অবজেক্ট তার নিজস্ব অ্যাট্রিবিউট ধারণ করবে।

আরেকটি কথা বলা প্রয়োজন। আমরা যে \_\_init\_\_ মেথড ব্যবহার করছি, যা অবজেক্ট তৈরি হওয়ার সময় আপনাআপনি কল হয়, একে বলে কনস্ট্রাকটর (constructor)।

### New way

**এখন ধরা যাক, Car ক্লাসের অবজেক্ট তৈরি করার পর আমার খেয়াল হলো, অবজেক্টের একটি ডেটা অ্যাট্রিবিউট দরকার, যেটি ক্লাসের মধ্যে নেই। আর আমি এখন ক্লাসটিও পরিবর্তন করতে চাইছি না। কিন্তু তাতে কোনো সমস্যা নেই। আমি চাইলে যেকোনো সময় আমাদের অবজেক্টের সঙ্গে একটি ডেটা অ্যাট্রিবিউট জুড়ে দিতে পারবো।**

**class Car:**

**def \_\_init\_\_(self, n, c):**

**self.name = n**

**self.color = c**

**def start(self):**

**print("Starting the engine")**

**car = Car("Corolla", "White")**

**car.year = 2017**

**print(car.name, car.color, car.year)**

## class Hello

class Hello:দিয়ে ক্লাসের নাম দিয়েছি। এরপর এর ভেতরে ক্লাসের ডেটা এবং মেথড গুলো। আমাদের Hello ক্লাসে একটি মাত্র মেথড রয়েছে। sayHello নামে। যাকে কল করলে Hello world! প্রিন্ট করবে।  
hello1 = Hello() এখানে Hello ক্লাসের একটি ইন্সট্যান্স তৈরি করে নিয়েছি, hello1 নামে। এরপর hello1.sayHello() দিয়ে Hello ক্লাসের sayHello মেথড টাকে কল করেছি। তা Hello World! প্রিন্ট করেছে। আমরা চাইলে Hello ক্লাসের একের অধিক ইন্সট্যান্স তৈরি করে নিতে পারি।

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | class Hello:       def sayHello(self):            print ("Hello world!")    hello1 = Hello()  hello1.sayHello()    hello2 = Hello()  hello2.sayHello() |

এখানে hello2 নামে আরেকটি ইন্সট্যান্স তৈরি করেছি। এরপর মেথড sayHello কল করেছি।

### Constractor

কনস্ট্রাকটর তৈরি করা হয় \_\_init\_\_ দিয়ে। দুইটা আন্ডারস্কোর init এবং দুইটা আন্ডারস্কোর দিয়ে। নিচের মত করে।

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | class Hello:       def \_\_init\_\_:       print (“Constractor")         def sayHello(self):            print ("Hello world!") |

যখন একটি অবজেক্ট তৈরি করা হয়, তখনই কনস্ট্রাকটর কল হয়। যত গুলো অবজেক্ট তৈরি করা হয়, তত বার কন্সট্রাকটর কল করা হয়ঃ

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | class Hello:       def \_\_init\_\_(self):            print ("Constractor")         def sayHello(self):            print ("Hello world!")    hello1 = Hello()  hello1.sayHello()    hello2 = Hello()  hello2.sayHello() |

কন্সট্রাকটর মূলত ডেটা সেট করার জন্য ব্যবহৃত হয়।

|  |
| --- |
| class Hello:       def \_\_init\_\_(self, name):            self.n = name    def sayHello(self):           return ("Hello " + self.n)    hello1 = Hello("Jack")    print (hello1.sayHello()) |

উপরের প্রোগ্রামে আমরা ক্লাসের ইন্সট্যান্স তৈরি করার সময় একটা নাম পাস করেছি। কনস্ট্রাকটরে অবজেক্ট থেকে পাস করা নামটি নিয়ে একটি ভ্যারিয়েবলে রাখা হয়েছে, self.n = name দিয়ে। এখনে self হচ্ছে ঐ অবজেক্টটি, যে অবজেক্ট আমরা ব্যবহার করতেছি। তারপর sayHello তে আমরা অবজেক্ট তৈরি করার সময় যে নাম দিয়েছি, Hello + ঐ নামটি রিটার্ন করেছি।

ক্লাস নিয়ে সহজ একটা উদাহরণ দিচ্ছি।

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | class Calculator:       def \_\_init\_\_(self, n):           self.number = n         def square(self):           return (self.number\*self.number)    calc = Calculator(4)  print (calc.square()) |

এখানে আমরা Calculator নামক একটা ক্লাস নিয়েছি। যার ভেতর একটি মেথড রয়েছে square নামে। যার কাজ হচ্ছে ক্লাসটির ইন্সট্যান্স তৈরির সময় কোন নাম্বার পাস করলে তার স্কয়ার রিটার্ন করা। স্কয়ার রিটার্ন করার মত করে আমরা যোগ, বিয়োগ, গুণ, ভাগের মেথড লিখতে পারি। পরে যোগ, বিয়োগ, গুণ ভাগ করতে চাইলে ঐ মেথড গুলো কল করলেই আমাদের যোগ, বিয়োগ করে দিবে।

এখন আমরা একটি Scientific Calculator তৈরি করতে চাই। সাইন্টিফিক ক্যালকুলেটরে যোগ বিয়োগ গুণ ভাগের পাশা পাশি আরো অনেক গুলো সুবিধে থাকে। যেমন একটা সংখ্যার সাইন মান বের করা। যেহেতু Calculator এ যোগ, বিয়োগ, গুণ ভাগ ইত্যাদি করা যায়, আমরা সাইন্টিফিক ক্যালকুলেটর তৈরির সময় আলাদা ভাবে ঐ Calculator ক্লাসের কোড গুলো না লিখে Calculator ক্লাসের সাথে আরো কিছু কোড যুক্ত করে দিলেই আমাদের কাজ হয়ে যাবে। আর এ যুক্ত করার জন্য আমরা নতুন একটা ক্লাস তৈরি করে নিব। SiCal = Scientific Calculator.

|  |
| --- |
| import math    class Calculator:       def \_\_init\_\_(self, n):            self.number = n         def square(self):            return (self.number\*self.number)    class SiCalc(Calculator):       def getSin(self):            return math.sin(self.number)    calc = SiCalc(45)    print (calc.getSin()) |

এখানে আমরা SiCalc নামে একটা নতুন ক্লাস তৈরি করেছি। যার বলে দিয়েছি এটির কোন প্যারেন্ট ক্লাস আছে কিনা। আমাদের SiClac ক্লাসের প্যারেন্ট ক্লাস বা মূল ক্লাস হছে Calculator. এটাকে বলে ডিরাইভ করা। যখন আমরা বেজ ক্লাস বা মূলক্লাস থেকে ডিরাইভ করে নতুন ক্লাস তৈরি করব, তখন আমরা বেজ ক্লাসের সকল মেথড ব্যবহার করতে পারব।

Example:

**import** math

class one:

**def** \_\_init\_\_(*self*,**x**):

*self*.x**=**x

**def** fun1(*self*):

        print(math.sin(*self*.x))

        print(math.cos(*self*.x))

        print(math.tan(*self*.x))

        print(math.sqrt(*self*.x))

lol**=**one(16)

lol.fun1()

example:

**import** math

class one:

**def** \_\_init\_\_(*self*,**x**):

*self*.x**=**x

**def** fun1(*self*):

        print(math.sin(*self*.x))

        print(math.cos(*self*.x))

        print(math.tan(*self*.x))

        print(math.sqrt(*self*.x))

cool**=**int(input('enter a intiger number: '))

lol**=**one(cool)

lol.fun1()

পাইথন অবজেক্ট অরিয়েন্টেড প্রোগ্রামিং  এ  অন্য সব ল্যাংগুয়েজের মত ক্লাস দিয়ে শুরু করতে হয়।

## অবজেক্ট অরিয়েন্টেড পাইথন

**class** **MyClass**:

**pass**

আমরা কিন্তু একটা ক্লাস তৈরি করে ফেললাম। এখন ক্লাসের ভিতরে আমরা ফাংশন লিখবো। উফ!!! সরি!!! ক্লাসের ভিতর ফাংশন কে বলে মেথড। তো আমরা ঐ একট মেথড তৈরি করি।

**class** **MyClass**: # আমরা এখানে ক্লাস তৈরি করেছি

**def** **\_\_init\_\_**(*self*, *name*, *age*):

self.name=name

self.age=age

এখানে আমরা একটি মেথড তেরি করলাম । অাসলে আমরা এখানে মেথডটি ইনিশিয়ালাইজ করলাম। আর এটি একটি স্বংক্রিয় মেথড। কারন একে আলাদা ভাবে কল করা লাগে না। আর আমরা এখানে self হল আইন্ডিটিফিয়ার। যদিও self এর বদলে অন্য কিছু লিখলেও সমস্যা নেই।  
  
তো আমরা এবার আমাদের ক্লাসের একটি নতুন ইনস্টান্স তৈরি করবো। আমরা a\_1 নামে একটি ইনস্টান্স তৈরি করলাম। আমরা এখন দুটি ভ্যালু পাস করলাম MyClass এর ভিতর।

**class** **MyClass**:

**def** **\_\_init\_\_**(*self*, *name*, *age*):

self.name=name

self.age=age

a\_1=MyClass('asad',23) # আমরা এখানে ইনস্টান্স তৈরি করলাম।

এখন যদি আমরা a\_1 ইনস্টান্স এর সাথে name টা প্রিন্ট করি তাহলে কি হবে? একটু দেখি!!!!!

**class** **MyClass**:

**def** **\_\_init\_\_**(*self*, *name*, *age*):

self.name=name

self.age=age

a\_1=MyClass('asad',23)

print(a\_1.name)

## ক্লাস

এরকম একটা গেম ডিজাইনে অবজেক্ট ওরিয়েন্টেড কনসেপ্ট খুবি উপকারী হতে পারে।

class কিওয়ার্ড ব্যবহার করে পাইথনে ক্লাস তৈরি করা হয়, একটি ক্লাসের মধ্যে বিভিন্ন মেথড(ফাংশন) এবং অ্যাট্রিবিউট (প্রপার্টি) থাকতে পারে যেগুলো পাইথনের নিয়ম অনুযায়ী ইন্ডেন্টেড ব্লকের মধ্যে থাকে।

*# The blueprint to create monsters*

class Monster:

**def** \_\_init\_\_(*self*, **color**, **heads**):

*self*.color **=** color

*self*.heads **=** heads

*# Create some real monsters*

fogthing **=** Monster("Black", 5)

mournsnake **=** Monster("Yellow", 4)

tangleface **=** Monster("Red", 3)

*# Check whether those monsters got different existence in memory or not*

print('I have ' **+** str(fogthing.heads) **+** ' heads and I\'m ' **+** fogthing.color)

print('I also have ' **+** str(mournsnake.heads) **+** ' heads and I\'m ' **+** mournsnake.color)

print('I got ' **+** str(tangleface.heads) **+** ' heads and I\'m ' **+** tangleface.color)

উপরে প্রথমেই একটি ক্লাস (আমাদের ভাষায় ব্লুএপ্রিন্ট বা টেম্পলেট) তৈরি করা হয়েছে। এর মধ্যে একটি ম্যাজিক মেথড আছে \_\_init\_\_ (যেটা নিয়ে নিচে আলোচনা আছে) এবং এর দুটো অ্যাট্রিবিউট আছে color ও heads এবং এই ক্লাসকে ব্যবহার করে বা ইন্সট্যান্টশিয়েট করে ৩টি অবজেক্ট তৈরি করা হয়েছে যেগুলো কিনা নিজেরা আলাদা আলাদা।

### \_\_init\_\_ **মেথড**

এই মেথডটি যেকোনো ক্লাসের খুবি গুরুত্বপূর্ণ একটি মেথড। যখনি কোন ক্লাস থেকে কোন অবজেক্ট বা ইন্সট্যান্স তৈরি করা হয় তখনি এই মেথডটি স্বয়ংক্রিয় ভাবে কল হয়। ক্লাসের মেথডের ক্ষেত্রে আরেকটি গুরুত্বপূর্ণ বিষয় হচ্ছে - সব মেথডের প্রথম প্যারামিটারটি হতে হয় self. যদিও এই মেথড গুলোকে কল করার সময় নির্দিষ্ট করে এই আর্গুমেন্টটি পাঠাতে হয় না (পাইথন নিজে থেকেই এটা ম্যানেজ করে)।

বস্তুত এই self, ওই মেথডকে কল করা ইন্সট্যান্সটিকেই নির্দেশ করে। অর্থাৎ, উপরের উদাহরণ অনুযায়ী যখন fogthing = Monster("Black", 5) লিখে fogthing নামের একটি অবজেক্ট তৈরি করা হচ্ছে। তখন কিন্তু \_\_init\_\_ কল হচ্ছে স্বয়ংক্রিয় ভাবে। আর এই \_\_init\_\_ মেথডের কাছে প্রথম আর্গুমেন্ট self হিসেবে চলে যাচ্ছে এই fogthing ইন্সট্যান্সটি। আর ওই মেথডের ডেফিনেশনের মধ্যে সেই self এর দুটো অ্যাট্রিবিউট color এবং heads কে সেট করা হচ্ছে (এর কাছে আসা দ্বিতীয় ও তৃতীয় আর্গুমেন্ট এর ভ্যালু নিয়ে)। তাহলে পক্ষান্তরে কিন্তু fogthing অবজেক্ট এর দুটো অ্যাট্রিবিউট সেট হয়ে গেলো অবজেক্ট তৈরির সাথে সাথেই।

dot চিহ্ন ব্যবহার করে একটি অবজেক্টের অ্যাট্রিবিউট এবং মেথড গুলোকে অ্যাক্সেস করা হয়। \_\_init\_\_ মেথড কে কন্সট্রাক্টর বলা হয়

### **ক্লাস অ্যাট্রিবিউট**

একটি ক্লাসের মধ্যে সাধারণ কিছু বৈশিষ্ট্য ডিফাইন করা যায় যেগুলো একটু অন্য রকম অর্থাৎ নির্দিষ্ট কোন অবজেক্টের বাধ্যগত না। যেমন, \_\_init\_\_ মেথডের সাহায্য নিয়ে, এর মধ্যেকার self কে কাজে লাগিয়ে আমরা একটি ক্লাস থেকে তৈরি করা অবজেক্টের কিছু অ্যাট্রিবিউট সেট করতে পারি যেগুলো ওই অবজেক্টের অ্যাট্রিবিউট। কিন্তু, একটি ক্লাসের এমন কিছু অ্যাট্রিবিউট থাকতে পারে যেগুলো এর সব অবজেক্টই অ্যাক্সেস করতে পারবে তথা সব অব্জেক্টেরই অ্যাট্রিবিউট বলা যেতে পারে। এদেরকে ক্লাস অ্যাট্রিবিউট বলা হয়।

class Monster:

    identity **=** "negative character"

**def** \_\_init\_\_(*self*, **color**, **heads**):

*self*.color **=** color

*self*.heads **=** heads

**def** attack(*self*):

        print("Just attacked a Hero, Mu...hahahaha!!!")

mournsnake **=** Monster("Yellow", 4)

tangleface **=** Monster("Red", 3)

print('I am a ' **+** str(mournsnake.heads) **+** ' headed ' **+** mournsnake.identity)

print('I am a ' **+** str(tangleface.heads) **+** ' headed ' **+** tangleface.identity)

উপরের উদাহরণে, Monster ক্লাসের একটি ক্লাস অ্যাট্রিবিউট আছে যার নাম identity এবং এটা এই ক্লাসের সব অবজেক্টের জন্যই এক এবং সব অবজেক্টই অ্যাক্সেস করতে পারে। আবার, যেহেতু এটা একটা ক্লাস অ্যাট্রিবিউট তাই একে ক্লাসের নাম দিয়েও অ্যাক্সেস করা যায় নিচের মত করে, print(Monster.identity) যার আউটপুট আসবে, negative character

### রিভিউ

ক্লাস হচ্ছে ব্লুপ্রিন্ট যার উপর ভিত্তি করে মেমোরিতে ওই টাইপের কিছু অবজেক্ট তৈরি করা যায়। class কিওয়ার্ড ব্যবহার করে ক্লাস তৈরি করা হয়। ক্লাসের কিছু নিজস্ব অ্যাট্রিবিউট থাকে যাদেরকে ক্লাস অ্যাট্রিবিউট বলে। এদেরকে ক্লাস অথবা ইন্সট্যান্স দিয়ে অ্যাক্সেস করা যায়। এছাড়া ক্লাসের কিছু ইন্সট্যান্স অ্যাট্রিবিউট থাকতে পারে যেগুলোকে ইন্সট্যান্স এর অ্যাট্রিবিউট বলা হয় এবং ইন্সট্যানন্স এর মাধ্যমেই অ্যাক্সেস করা হয়। \_\_init\_\_ মেথড এর সাহায্যে যেকোনো ক্লাস থেকে অবজেক্ট তৈরির সময় ব্যাসিক কিছু কাজ করে নেয়া যায়। এছাড়া ক্লাসের অনেক কাস্টম মেথড থাকতে পারে যেগুলো পক্ষান্তরে ওই ক্লাস থেকে তৈরি অবজেক্ট গুলোরই মেথড হিসেবে ব্যবহার করা যায়।

## অবজেক্ট অরিয়েন্টেড প্রোগ্রামিং এর তিনটি নীতি বা তিনটি প্রধান বৈশিষ্ট।

সকল অবজেক্ট অরিয়েন্টেড প্রোগ্রামিং এর কোর তিনটি মৈলিক বৈশিষ্ট রয়েছে। সেগুলো হচ্ছেঃ

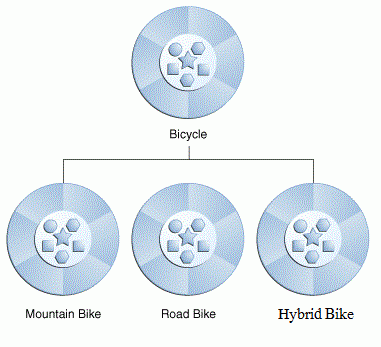
* Encapsulation
* Inheritance
* Polymorphism

Encapsulation হচ্ছে জাভা বা যেকোন অবজেক্ট অরিয়েন্টেড প্রোগ্রামের প্রধান বা মৈলিক বৈশিষ্ট।

ফেসবুকের প্রাইভেসি নিয়ে একটু আলোচনা করা যাক। আচ্ছা, আপনি আপনার প্রোফাইলে যেসব সুবিধে পাবেন তা অন্য কারো প্রোফাইলে পাবেন না। কিছু জিনিস থাকে পাবলিক, কিছু জিনিস থাকে প্রাইভেট আবার কিছু জিনিস থাকে শুধু আপনার বন্ধুরা দেখবে তাই না? সব কিছু যদি সবাই দেখতে পারত তাহলে কি অসুবিধেই না হতো তাই না?

তাহলে Encapsulation মানে কি তাই না?Encapsulation মানে হচ্ছে মোড়ানে, সব কিছুকে একটা প্যাকেটের মধ্যে ঢুকানো। প্রোগ্রামে এটা ব্যবহার করা হয় ডাটা লুকানোর জন্য। যে ডাটা গুলো পাবলিক যে গুলো যেন সবাই দেখতে পারে, যে গুলো প্রাইভেট সে গুলো যেন ঐ ডাটার মালিক ছাড়া আর কেউ না দেখতে পারে, এটা হচ্ছে Encapsulation  এর কাজ।

Inheritance হচ্ছে একটা অবজেক্ট আরেকটা অবজেক্ট এর বৈশিষ্ঠ অর্জন করার প্রক্রিয়া।

প্রত্যেকটি ক্লাসের কিছু আট্রিবিউট থাকে আর কিছু মেথড থাকে।  
বাই-চাইকেল এর কথা চিন্তা করি। চাইকেল গুলো অনেক রকম হয়, যেমন রোড বাইক, মাউন্টেন বাইক বা হাইব্রিড বাইক। এখন সব গুলো চাইকেল এর ই তো চাকা থাকে, টায়ার থাকে, প্যাডেল থাকে, ব্রেক থাকে ইত্যাদি ইত্যাদি।  
[](https://i1.wp.com/jakir.me/wp-content/uploads/2013/05/inheritance-bike-.gif)  
একটার সাথে আরেকটার কিছু ফাংশনাল পার্থ্যক্য রয়েছে শুধু।  ধরে নি আমরা চাইকেলের প্রোগ্রাম লিখব। প্রোগ্রামটি সহজে লেখার জন্য আমরা বার বার সব ধরনের আর্টিবিউট আর মেথড গুলো না লিখে জেনেরিক বা কমন মেথড গুলো এক সাথে একটা ক্লাসে লিখব। যার নাম হচ্ছে Bicycle. এবার যখন রোড বাইকের ক্লাস লিখব, তখন পুনরায় Bicycle এর বা কমন আর্টিবিউট আর মেথড গুলো না লিখে সে গুলোকে Bicycle ক্লাস থেকে নিয়ে আসব। আর এ প্রক্রিয়াই হচ্ছ ইনহেরিটেন্স।

Polymorphism সম্পর্কে জানার আগে এর ইংরেজী মিনিং কি তা বের করি কারন এটি একটি গীক শব্দ। Polymorphism ভাংলে পাওয়া যায় many (poly) shapes (morph) যার মানে হচ্ছে বহুরূপতা। প্রোগ্রামিং এ ও একই অর্থ বহন করে। একই জিনিসের বহু আকৃতি বা গঠনই হচ্ছে Polymorphism, এটি আসলেই অনেক ভালো একটা বৈশিষ্ট। ভালো মানের পোগ্রাম লেখার সময় অনেক ভালো কাজ করে। আর কোডের পরিমান ও কমিয়ে দেয়।

## ইনহেরিট্যান্স

উত্তরাধিকার সূত্রে প্রাপ্ত কোনো কিছুকে ইনহেরিট্যান্স বলে। বাপ ভালো রাষ্ট্রনেতা ছিল, তাই মেয়েও ভালো রাষ্ট্রনেতা হয়েছে- এটা হলো ইনহেরিট্যান্সের সফলতম উদাহরণ। এখানে গুণটা উত্তরাধিকার সূত্রে প্রাপ্ত।

আমরা আমাদের প্রোগ্রামিং চিন্তা-চেতনায় ফেরত আসি। আগের চ্যাপ্টারে আমরা ক্যালকুলেটর নামে একটা ক্লাস বানিয়েছিলাম। এখন আমরা একটা সুপারক্যালকুলেটর ক্লাস বানাব। যেখানে আগের Calculator ক্লাসের সব মেথড তো থাকবেই, তার সাথে এক্সট্রা কয়েকটা মেথড যেমন- স্কয়ার বের করা, কিউব বের করাও থাকবে। তাহলে এতক্ষণ অবধি অর্জিত বিদ্যা দিয়ে নিচের মতো করে আমরা প্রোগ্রামটা লিখতে পারি -

class SuperCalculator:

"""Do addition, subtraction, multiplication, division, square and cube."""

def addition(self, a, b):

return a + b

def subtraction(self, a, b):

return a - b

def multiplication(self, a, b):

return a \* b

def division(self, a, b):

try:

return a / b

except ZeroDivisionError:

return 'It is impossible to divide by zero.'

def square(self, a):

return a \* a

def cube(self, a):

return a \* a \* a

my\_calculator = SuperCalculator()

temp = my\_calculator.addition(23, 47)

print(temp)

temp = my\_calculator.subtraction(87, 54)

print(temp)

temp = my\_calculator.multiplication(65, 56)

print(temp)

temp = my\_calculator.division(852, 76)

print(temp)

temp = my\_calculator.square(7)

print(temp)

temp = my\_calculator.cube(3)

print(temp)

**আউটপুট**

70

33

3640

11.210526315789474

49

27

অনেক কষ্ট করলাম আমরা। আচ্ছা আমাদের Calculator ক্লাসটার উপর ভিত্তি করে কোনকিছু করা গেলে কাজটা আরো সহজে হত না? ঐ ক্লাসে তো চারটা মেথড লেখাই আছে আমাদের। ওটার সাথে আর দুইটা মেথড যোগ করলেই কিন্তু কেল্লা ফতে!

class Calculator:

"""Do addition, subtraction, multiplication and division."""

def addition(self, a, b):

return a + b

def subtraction(self, a, b):

return a - b

def multiplication(self, a, b):

return a \* b

def division(self, a, b):

try:

return a / b

except ZeroDivisionError:

return 'It is impossible to divide by zero.'

class SuperCalculator(Calculator):

"""Child class of Calculator. Do square and cube."""

def square(self, a):

return a \* a

def cube(self, a):

return a \* a \* a

my\_calculator = SuperCalculator()

temp = my\_calculator.addition(23, 47)

print(temp)

temp = my\_calculator.subtraction(87, 54)

print(temp)

temp = my\_calculator.multiplication(65, 56)

print(temp)

temp = my\_calculator.division(852, 76)

print(temp)

temp = my\_calculator.square(7)

print(temp)

temp = my\_calculator.cube(3)

print(temp)

**আউটপুট**

70

33

3640

11.210526315789474

49

27

এখানে আমরা Calculator ক্লাসের উপর বেইস করে বা Calculator ক্লাসকে ইনহেরিট করে SuperCalculator নামে নতুন একটা ক্লাস তৈরি করেছি। নতুন ক্লাসের নামের পর () এর ভিতর যে ক্লাসগুলো থেকে ইনহেরিট করা হবে তাদের কমা সেপারেটেড লিস্ট দিতে হয়। আমরা শুধু একটা ক্লাস থেকে ইনহেরিট করেছি তাই শুধু একটা ক্লাসকে দিয়েছি। এই ক্লাসকে বেইস (base) ক্লাস বা প্যারেন্ট (parent) ক্লাস বলে। আর নতুন তৈরি ক্লাসকে বলে চাইল্ড (child) ক্লাস। এই চাইল্ড ক্লাসে প্যারেন্ট ক্লাসের সকল গুণাগুণ অক্ষুণ্ন থাকে। চাইল্ড ক্লাসের ইন্সট্যান্স বা অবজেক্ট থেকে প্যারেন্ট ক্লাসের সকল মেথড অ্যাক্সেস করা যায়।

Example:

class one:

**def** \_\_init\_\_(*self*,**x**,**y**):

*self*.x**=**x

*self*.y**=**y

**pass**

**def** sum(*self*):

**return** *self*.x**+***self*.y

**def** sub(*self*):

**return** *self*.x**-***self*.y

**def** mul(*self*):

**return** *self*.x**\****self*.y

class two(*one*):

**def** div(*self*):

**return** *self*.x**/***self*.y

class three(*two*):

**def** squre(*self*):

**return** *self*.x**\****self*.x

lol**=**three(2,2)

print(lol.sum())

print(lol.sub())

print(lol.mul())

print(lol.div())

print(lol.squre())

যদিও আপাত দৃষ্টিতে মনে হচ্ছে ইনহেরিট করে কোন লাভ হয়নি। তবে সত্যি বলতে কি, লাভ হয়েছে। নানা সময়ে পাইথনে, বিভিন্ন ক্লাস ইনহেরিট করে আমাদের কাজ করতে হবে। যেমনটা করতে হয় ইউজার ডিফাইন্ড এক্সেপশন পয়দা করার সময়। এরর হ্যান্ডলিং চাপ্টারে আমরা বলেছিলাম পরে একসময় আমরা এই জিনিসটা শিখব। এখন সে সময় উপস্থিত। পাইথনে ইউজার ডিফাইন্ড এক্সেপশন প্রত্যক্ষ বা পরোক্ষভাবে Exception ক্লাস থেকে ডিরাইভ (derive) করতে হবে।

class CustomError(Exception):

"""Just for example."""

def \_\_init\_\_(self, message):

self.message = message

try:

raise CustomError('It is a Custom Error.')

except CustomError as e:

print(e)

**আউটপুট**

It is a Custom Error.

### মেথড ওভাররাইডিং

মেথড ওভাররাইডিং (overriding) একটা চমৎকার জিনিস। চাইল্ড ক্লাসে আমরা প্যারেন্ট ক্লাসের মেথডকে ওভাররাইড করে ঢেলে সাজাতে পারি। একেই মেথড ওভাররাইডিং বলে। একটা উদাহরণ দেখা যাক।

class Calculator:

"""Do addition, subtraction, multiplication and division."""

def addition(self, a, b):

return a + b

def subtraction(self, a, b):

return a - b

def multiplication(self, a, b):

return a \* b

def division(self, a, b):

try:

return a / b

except ZeroDivisionError:

return 'It is impossible to divide by zero.'

class SuperCalculator(Calculator):

"""Do addition, subtraction, multiplication, division, square and cube."""

def addition(self, a, b, c):

return a + b + c

def square(self, a):

return a \* a

def cube(self, a):

return a \* a \* a

my\_calculator = SuperCalculator()

temp = my\_calculator.addition(23, 47, 12)

print(temp)

temp = my\_calculator.subtraction(87, 54)

print(temp)

temp = my\_calculator.multiplication(65, 56)

print(temp)

temp = my\_calculator.division(852, 76)

print(temp)

temp = my\_calculator.square(7)

print(temp)

temp = my\_calculator.cube(3)

print(temp)

**আউটপুট**

82

33

3640

11.210526315789474

49

27

এখানে আমরা addition() মেথডকে ওভাররাইড করেছি। প্যারেন্ট ক্লাসে এই মেথড কেবল দুইটা সংখ্যাকে যোগ করতে পারত। আর চাইল্ড ক্লাসে এই মেথড তিনটা সংখ্যাকে যোগ করতে পারে। এই একই ভাবে ক্লাস কন্সট্রাক্টর \_\_init\_\_() কেও আমরা ওভাররাইড করতে পারি।

এবার বাড়ির কাজ। ক্লাস কন্সট্রাক্টর \_\_init\_\_() কে ওভাররাইড করে একটা সুপারক্যালকুলেটর ক্লাস বানিয়ে প্রোগ্রাম লিখতে হবে। পারা যাবে তো? অবশ্যই। কারণ, মানুষ আমাদের পাশে আছে।

Example:

class one:

**def** \_\_init\_\_(*self*,**x**,**y**):

*self*.x**=**x

*self*.y**=**y

**pass**

**def** sum(*self*):

**return** *self*.x**+***self*.y

**def** sub(*self*):

**return** *self*.x**-***self*.y

**def** mul(*self*):

**return** *self*.x**\****self*.y

class two(*one*):

**def** div(*self*):

**return** *self*.x**/***self*.y

class three(*two*):

**def** squre(*self*):

**return** *self*.x**\****self*.x

**def** sum(*self*):

**return** *self*.x**+***self*.y**-**10

lol**=**three(2,2)

print(lol.sum())

print(lol.sub())

print(lol.mul())

print(lol.div())

print(lol.squre())

class Monster:

**def** \_\_init\_\_(*self*, **name**, **color**):

*self*.name **=** name

*self*.color **=** color

**def** attack(*self*):

        print('I am attacking...')

class Fogthing(*Monster*):

**def** make\_sound(*self*):

        print('Grrrrrrrrrr\n')

class Mournsnake(*Monster*):

**def** make\_sound(*self*):

        print('Hiiissssshhhh\n')

fogthing **=** Fogthing("Fogthing", "Yellow")

fogthing.attack()

fogthing.make\_sound()

mournsnake **=** Mournsnake("Mournsnake", "Red")

mournsnake.attack()

mournsnake.make\_sound()

Hiiissssshhhh

কোন ক্লাসকে ইনহেরিট করার জন্য ওই ক্লাসের নামটি নতুন ক্লাসের নামের পর ব্র্যাকেটের মধ্যে লিখতে হয়। এখানে Monster হচ্ছে সুপারক্লাস আর Fogthing, Mournsnake হচ্ছে সাবক্লাস

**অভাররাইড** যদি সুপারক্লাসের কোন মেথড বা অ্যাট্রিবিউটকে এর একটা সাবক্লাসের মধ্যে আবার ডিফাইন করা হয় তাহলে সেগুলো অভাররাইড হয়ে যায়। যেমন,

class Monster:

**def** \_\_init\_\_(*self*, **name**, **color**):

*self*.name **=** name

*self*.color **=** color

**def** attack(*self*):

        print('I am attacking...')

class Fogthing(*Monster*):

**def** attack(*self*):

        print('I am killing...')

**def** make\_sound(*self*):

        print('Grrrrrrrrrr\n')

fogthing **=** Fogthing("Fogthing", "Yellow")

fogthing.attack()

fogthing.make\_sound()

উপরের উদাহরণে, Fogthing ক্লাস মুল Monster ক্লাসের attack মেথডকে অভাররাইড করেছে।

### **মাল্টিপল ইনহেরিট্যান্স**

নিচের প্রোগ্রামটি দেখি এবং অনুমান করি এর আউটপুট কি আসবে,

class A:

**def** where(*self*):

        print("I am from class A")

class B:

**def** where(*self*):

        print("I am from class B")

class C(*A*, *B*):

**pass**

an\_instance\_of\_c **=** C()

an\_instance\_of\_c.where()

এখানে, C ক্লাসটি A এবং B দুটো ক্লাসকেই ইনহেরিট করেছে। আবার ওই দুটো ক্লাসের প্রত্যেকটিতেই where নামের মেথড আছে। পরিশেষে, যখন C ক্লাসের ইন্সট্যান্স তৈরি করে where মেথডকে কল করা হচ্ছে তখন আসলে কোন ক্লাসের where মেথডটি কল হবে? প্রথমেই সেই মেথডকে C ক্লাসের মধ্যে খোঁজা হবে, না পেলে A ক্লাসের মধ্যে আর সেখানেও না পেলে B ক্লাসের মধ্যে খোঁজা হবে।

এই অর্ডার ভিত্তিক মেথড খোঁজার জন্য পাইথন নিজস্ব নিয়ম ফলো করে এবং সেটা দেখতে চাইলে আমরা প্রোগ্রামে print(C.mro()) স্টেটমেন্টটি ব্যবহার করতে পারি যার আউটপুট আসবে নিচের মত,

### super **মেথড**

ইনহেরিট্যান্স এর ক্ষেত্রে super একটি গুরুত্বপূর্ণ মেথড। এর মাধ্যমে একটি অবজেক্টের সুপার ক্লাসের মধ্যেকার মেথডকে কল করা যায়। যেমন,

class A:

**def** spam(*self*):

        print(1)

class B(*A*):

**def** spam(*self*):

        print(2)

        super().spam()

B().spam()

Example:

*#use super alter formula in method overriding*

class one:

**def** \_\_init\_\_(*self*):

        print('i am init method 1')

**def** fun(*self*):

        print('i am fun1')

class two:

**def** \_\_init\_\_(*self*):

        print('i am init metod 2')

**def** fun(*self*):

        print('i am fun2')

class three(*one*,*two*):

**def** \_\_init\_\_(*self*):

        one.\_\_init\_\_(*self*)

        two.\_\_init\_\_(*self*)

        print('i am init method 3')

**def** fun(*self*):

        one.fun(*self*)

        two.fun(*self*)

        print('i am fun3')

lol**=**three()

lol.fun()

example:

*#use 'super' to call all class method from last inharitence call*

class one:

**def** fun1(*self*):

        print('i am fun1')

class two(*one*):

**def** fun2(*self*):

        print('i am fun2')

class three(*two*):

**def** fun3(*self*):

        print('i am fun3')

class four(*three*):

**def** fun4(*self*):

        super().fun1()

        super().fun2()

        super().fun3()

        print('i am fun4')

lol**=**four()

lol.fun4()

class MyList(list):

def \_\_setitem\_\_(self, index, value):

if index == 0:

raise IndexError

if index > 0:

index -= 1

super().\_\_setitem\_\_(index, value)

def \_\_getitem\_\_(self, index):

if index == 0:

raise IndexError

if index > 0:

index -= 1

return super().\_\_getitem\_\_(index)

mylist = MyList(['a', 'b', 'c', 'd', 'e'])

mylist[1] = 'apple'

mylist[5] = 'orange'

print(mylist)

print(mylist[1])

print(mylist[5])

MyList এ list এর দুটি ম্যাজিক মেথড \_\_setitem\_\_ এবং \_\_getitem\_\_ অল্টার করেছি। আমরা যখন alist[2] = 5 এরকম কোড লিখি তখন \_\_setitem\_\_ মেথড টি কল হয়। আবার যখন value = alist[2] এরকম কোড লিখি তখন \_\_getitem\_\_ মেথড টি কল হয়।

এখন আমাদের চাহিদা অনুযায়ী index শুরু হতে হবে 1 থেকে। তাই \_\_setitem\_\_ এবং \_\_getitem\_\_ এর ভেতর ওরকম লজিক লিখেছি। এবং তার পর প্যারেন্ট ক্লাস থেকে সংশ্লিষ্ট মেথড কে কল করেছি (অল্টার করেছি)।

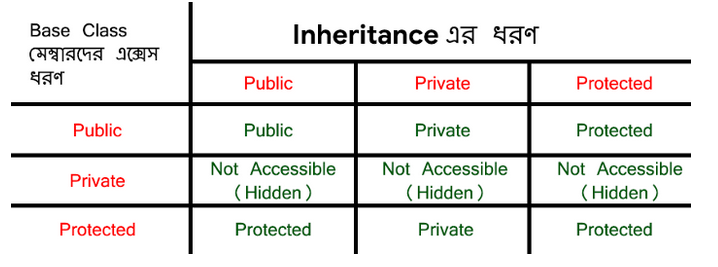
### **ইনহেরিটেন্সের বিভিন্ন Modes**

১. **Public Mode:** আমরা যদি পাবলিক base-class থেকে sub-class এ ডেটা ইনহেরিট করি তাহলে base-class এর পাবলিক মেম্বার গুলো sub-class এ পাবলিক থাকবে এবং প্রোটেকটেড মেম্বার গুলো প্রোটেকটেড থাকবে।

২. **Private Mode:** প্রাইভেট base-class থেকে sub-class এ ডেটা ইনহেরিট করি তাহলে base-class এর পাবলিক এবং প্রোটেকটেড দুটো মেম্বারই sub-class এ প্রাইভেট হয়ে যাবে।

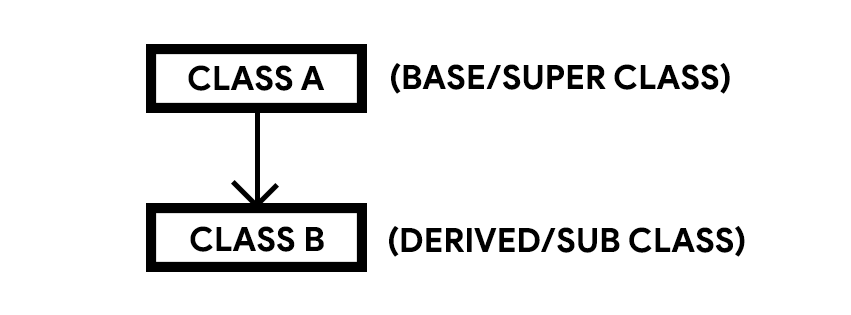
৩. **Protected Mode:** প্রোটেকটেড base-class থেকে sub-class এ ডেটা ইনহেরিট করি তাহলে base-class এর পাবলিক এবং প্রোটেকটেড দুটো মেম্বারই sub-class এ প্রোটেকটেড হয়ে যাবে।

base-class এর প্রাইভেট মেম্বার গুলোকে sub-class থেকে সরাসরি এক্সেস করা যায় না, কিন্তু প্রোটেকটেড মেম্বার গুলোকে এক্সেস করা যায়

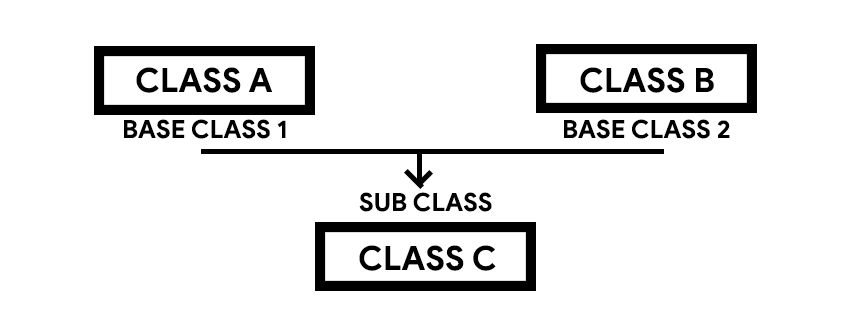


### **Inheritance এর প্রকারভেদ**

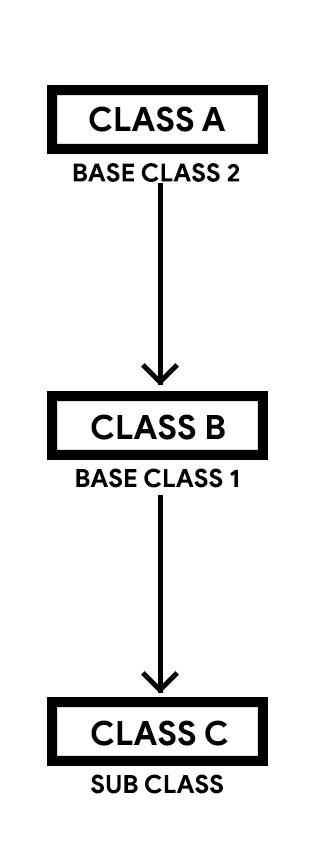
১. **Single Inheritance —** একটি class শুধু মাত্র অন্য আরেকটি class থেকে ইনহেরিট করতে পারবে। অর্থাৎ একটি sub-class শুধু মাত্র একটি base-class থেকে ইনহেরিট করতে পারবে।



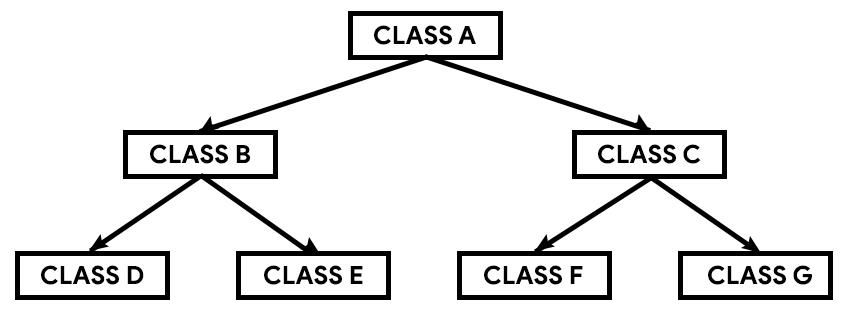
২. **Multiple Inheritance** — এটা হচ্ছে সি++ এর স্পেশাল ফিচার, যেটার মাধ্যমে একটি sub-class একাধিক base-class থেকে ইনহেরিট করতে পারবে।



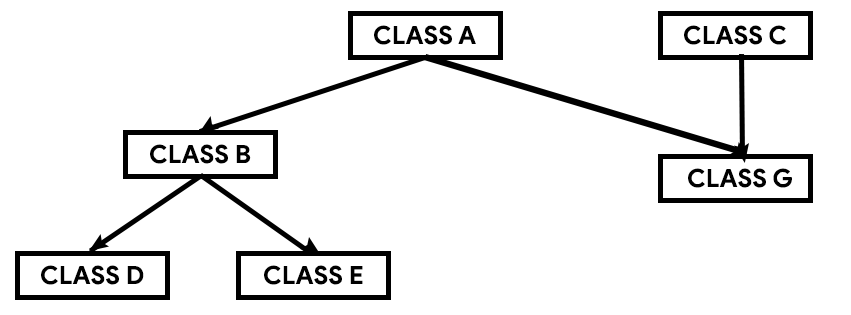
**৩. Multilevel Inheritance —** একটি sub class অথবা derived class থেকে আরেকটি sub class অথবা derived class কে তৈরি করা।



**৪. Hierarchical Inheritance —** এই ধরনের ইনহেরিটেন্সে একটি base-class থেকে একাধিক sub class অথবা derived class কে তৈরি করা যায়।



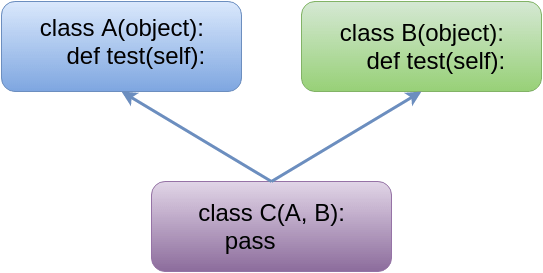
**৫. Hybrid (Virtual) Inheritance —** একাধিক ধরনের ইনহেরিটেন্স সিস্টেম মিলে Hybrid (Virtual) Inheritance তৈরি করা হয়। যেমন Hierarchical এবং Multiple Inheritance মিলে একটি Hybrid (Virtual) Inheritance তৈরি করা যায়।



### পাইথনে মাল্টিপল ইনহ্যারিটেন্স যেভাবে কাজ করে

বেশির ভাগ প্রোগ্রামারই মাল্টিপল ইনহ্যারিটেন্স ব্যাবহার করতে নিষেধ করেন। তারপরও বিশেষ কিছু ক্ষেত্রে আমাদের মাল্টিপল ইনহ্যারিটেন্স ব্যাবহার করতে হয়। তাহলে জেনে নেয়া যাক পাইথনে মাল্টিপল ইনহ্যারিটেন্স যে অর্ডার অনুসারে কাজ করে।

নিচের ডায়াগ্রাম টি দেখি:

[](https://i0.wp.com/fazle.me/wp-content/uploads/2016/06/Python-Multiple-Inheritance-1.png?ssl=1)

এই ডায়াগ্রাম অনুযায়ী কোড লিখি তাহলে:

class A:

**def** test(*self*):

        print("Test from A")

class B:

**def** test(*self*):

        print("Test from B")

class C(*A*, *B*):

**pass**

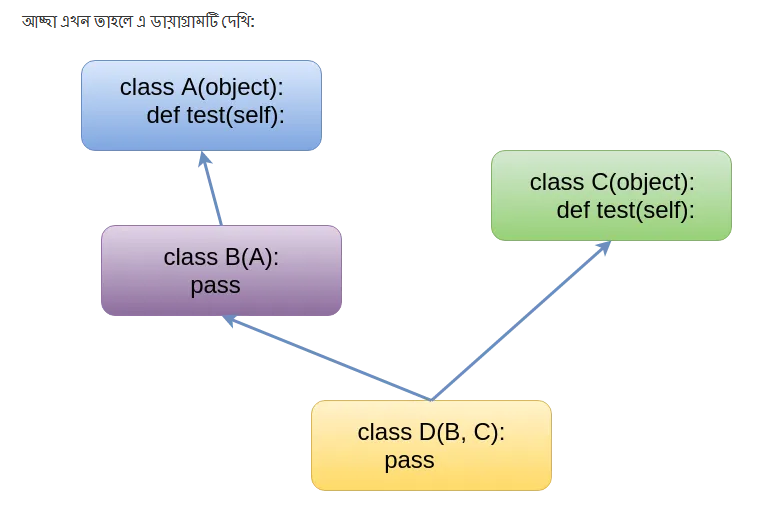
c\_instance **=** C()

c\_instance.test()

অর্থাৎ আমরা যখন c\_instance এর উপর test() মেথডটি কল করলাম, পাইথন প্রথমে এ মেথড টিকে C ক্লাসে খুজেছে। না পেয়ে এরপর A ক্লাসে খুজেছে। এখানে পেয়ে গেছে। এখানেও না পেলে তখন B ক্লাসে খুঁজত।

এই যে খোঁজাখুজি, পাইথন কোনটার পর কোনটায় খুঁজবে – এ ব্যাপারটিকে বলে মেথড রেজ্যুলেশন অর্ডার। আমরা যদি C ক্লাস এর মেথড রেজুলেশন অর্ডার দেখতে চাই, তাহলে যা করতে হবে:

print(C.mro())



class A:

**def** test(*self*):

        print("Test from A")

class B(*A*):

**pass**

class C:

**def** test(*self*):

        print("Test from C")

class D(*B*, *C*):

**pass**

lol **=** D()

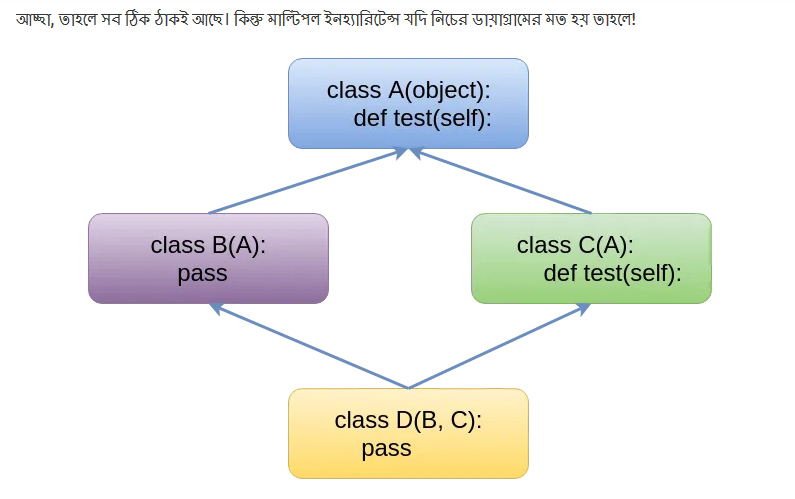
lol.test()

কি! উল্টাপাল্টা মনে হচ্ছে?

আসলে পাইথনে মেথড রেজ্যুলেশন অর্ডার [ডেপথ-ফার্স্ট সার্চিং](https://en.wikipedia.org/wiki/Depth-first_search) উপায়ে কাজ করে। ডেপথ-ফার্স্ট অনুযায়ী তাহলে অর্ডার হবে এমন: D -> B -> A -> C। তাই আমাদের আউটপুট এমন এসেছে।

আরেকটু নিশ্চিত হই:

print(D.mro())



class A:

**def** test(*self*):

        print("Test from A")

class B(*A*):

**pass**

class C(*A*):

**def** test(*self*):

        print("Test from C")

class D(*B*, *C*):

**pass**

lol **=** D()

lol.test()

এই কোডের আউটপুট কী হব?

আউটপুট জানার আগে D ক্লাসের মেথড রেজ্যুলেশন অর্ডার বের করার চেষ্টা করি। ডেপথ-ফার্স্ট অনুযায়ী মেথড রেজ্যুলেশন অর্ডার এরকম হবার কথা: D -> B -> A -> C -> A।

তাহলে আউটপুট আসার কথা:

Test from A

এখন কোডটি রান করে দেখি কী আউটপুট আসে:

Test from C

আহা! এরকম আউটপুট আসল কী করে! মেথড রেজ্যুলেশন অর্ডার চেক করে দেখি তো:

|  |  |
| --- | --- |
| 21 | print(D.mro()) |

আউটপুট:

[<class '\_\_main\_\_.D'>, <class '\_\_main\_\_.B'>, <class '\_\_main\_\_.C'>, <class '\_\_main\_\_.A'>, <class 'object'>]

অর্থাৎ D -> B -> C -> A। তাহলে এটা ডেপথ-ফার্স্ট হলো কী করে!

পাইথন ২.৩ থেকে নতুন একটি রুল যুক্ত হয়েছে। মেথড রেজ্যুলেশন অর্ডারের ডেপথ-ফার্স্টে যদি একই উপাদান একাধিক বার আসে, তাহলে একেবারে শেষটি রেখে বাকি গুলো মুছে দেয়া হয়। এখানে A দুইবার এসেছিল। তাই শুধু শেষেটি রাখা হয়েছে।

### ম্যাজিক মেথড

পাইথনে কিছু বিশেষ ধরনের বিল্ট ইন মেথড আছে যেগুলোকে ম্যাজিক মেথড বলা হয়। এগুলার চেনার খুব সহজ উপায় হচ্ছে এদের নামের দুই পাসেই দুটো করে আন্ডারস্কোর সিম্বল থাকে। অর্থাৎ, \_\_init\_\_ মেথডের মত। এই মেথডের সঙ্গে ইতোমধ্যে আমাদের পরিচয় হয়েছে। কোন ক্লাসে এই ম্যাজিক মেথড ব্যবহার করলে এবং পরবর্তীতে সেই ক্লাসের ইন্সট্যান্স তৈরির সময় এই ম্যাজিক মেথডটি স্বয়ংক্রিয় ভাবেই কল হয় যাতে করে এর মাধ্যমে কিছু সেটআপ রিলেটেড কাজ করে নেয়া যায়।

এই মেথড গুলোকে ভাষায় প্রকাশ করার সময় একটু জটিলতা হয়। যেমন, "আন্ডারস্কোর আন্ডারস্কোর ইনিট আন্ডারস্কোর আন্ডারস্কোর" এভাবে বললে অদ্ভুত শোনায়। তাই এদেরকে সুন্দর ভাবে "dunders" তথা "ডাণ্ডার ইনিট" এভাবে বলা হয়ে থাকে।

তো, এই \_\_init\_\_ মেথড বাদেও অনেক গুলো ম্যাজিক মেথড আছে পাইথনে। ম্যাজিক মেথডের খুব বহুল ব্যবহার দেখা যায় অপারেটর অভারলোডিং এর সময় যা পরের চ্যাপ্টারেই আলোচনা করা হয়েছে। প্রত্যেকটি অপারেটর এর জন্যই একটি ম্যাজিক মেথড আছে। যেমন, + অপারেটর এর জন্য ম্যাজিক মেথডটি হচ্ছে \_\_add\_\_

ঘটনাটি এভাবে ঘটে, যদি আমাদের এমন একটি এক্সপ্রেশন থাকে x+y এবং x বস্তুত K ক্লাসের ইন্সট্যান্স হয়। তখন পাইথন K ক্লাসের ডেফিনেশন চেক করবে। যদি K ক্লাসের একটি মেথড থাকে \_\_add\_\_ তাহলে সেটাকে কল করা হবে এভাবেঃ x.\_\_add\_\_(y)

কিছু কমন অপারেটরের ম্যাজিক মেথডঃ \_\_sub\_\_ হচ্ছে - জন্য \_\_mul\_\_ হচ্ছে \* জন্য \_\_truediv\_\_ হচ্ছে / জন্য \_\_floordiv\_\_ হচ্ছে // জন্য \_\_mod\_\_ হচ্ছে % জন্য \_\_pow\_\_ হচ্ছে \*\* জন্য \_\_and\_\_ হচ্ছে & জন্য \_\_xor\_\_ হচ্ছে ^ জন্য \_\_or\_\_ হচ্ছে | জন্য

তুলনা করার অপারেটর গুলোর জন্যও পাইথনে ম্যাজিক মেথড আছেঃ \_\_lt\_\_ হচ্ছে < জন্য \_\_le\_\_ হচ্ছে <= জন্য \_\_eq\_\_ হচ্ছে == জন্য \_\_ne\_\_ হচ্ছে != জন্য \_\_gt\_\_ হচ্ছে > জন্য \_\_ge\_\_ হচ্ছে >= জন্য

এছাড়াও আরও অনেক ম্যাজিক মেথড আছে পাইথনে যেমন - \_\_len\_\_ \_\_getitem\_\_ \_\_setitem\_\_ \_\_delitem\_\_ \_\_iter\_\_ \_\_contains\_\_ ইত্যাদি

### অপারেটর অভারলোডিং

আমরা আগের চ্যাপ্টারগুলোতে নানাবিধ অপারেশন দেখেছি - যোগ, বিয়োগ, গুন ভাগ ইত্যাদি । পাইথনের একটা চমৎকার ফিচার হচ্ছে এই অপারেটরগুলোর ফাংশনালিটি পরিবর্তন করা যায় ।

প্রথমেই আমরা দেখে নেই, এই অপারেটর গুলো আসলে কিভাবে কাজ করে । আমরা যখন কোন অপারেটর ব্যবহার করি, যেমন:

a = b + c

পাইথন এই + অপারেশনের ফলাফল নির্নয়ের জন্য ইন্টারনালি b অবজেক্টের \_\_add\_\_ মেথডে আর্গুমেন্ট হিসেবে c কে পাস করে দেয় । ঐ মেথডের রিটার্ন ভ্যালুই হয় উক্ত অপারেশনের ফলাফল । অর্থাৎ, উপরে দেখানো অপারেশনটি আসলে এভাবে কাজ করে -

a = b.\_\_add\_\_(c)

এখানে লক্ষ্য করুন, আপনি যদি এখন এই b অবজেক্টের \_\_add\_\_ মেথডটি পরিবর্তন করে দেন, তাহলে ঐ b অবজেক্টের উপর + অপারেশনের ফাংশনালিটিও পরিবর্তন হয়ে যাচ্ছে ।

এই জিনিসটাই হচ্ছে অপারটের ওভারলোডিং ।

আসুন একটি উদাহরন দেখে নেইঃ

### ম্যাজিক মেথড

ক্লাস নিয়ে পড়াশুনা করতে গিয়ে আমরা \_\_init\_\_(), \_\_iter\_\_(), \_\_next\_\_() এরকম কিছু মেথড দেখেছি। এই তিনটা বাদেও পাইথনে ডাবল আন্ডারস্কোর ঘেরা আরো অনেক স্পেশাল মেথড রয়েছে। এদেরকে ম্যাজিক মেথড বলে। কারণ এরা ক্লাসের কাজে-কর্মে স্রেফ ম্যাজিক নিয়ে আসে। আগে-পিছে দুইটা আন্ডারস্কোর থাকলে তাকে ডান্ডার (dunder) বলে। আর পড়ার সময় আন্ডারস্কোর আন্ডারস্কোর ইনিট আন্ডারস্কোর আন্ডারস্কোর এভাবে না বলে ‘ডান্ডার ইনিট’ এভাবে পড়া হয়।

পাইথনের অফিসিয়াল ডকে ( <https://docs.python.org/3/reference/datamodel.html> ) আমরা ম্যাজিক মেথড সম্পর্কে বিস্তারিত জানতে পারি। তবে এখানে আমরা গুরুত্বপূর্ণ কিছু ম্যাজিক মেথড সম্পর্কে জানব।

#### কন্সট্রাকশন ও ডেস্ট্রাকশন ম্যাজিক মেথড

| **মেথড** | **বর্ণনা** |
| --- | --- |
| \_\_init\_\_(self, [...]) | ক্লাস কন্সট্রাক্ট করে। |
| \_\_del\_\_(self) | ক্লাস ডেস্ট্রাক্ট করে। |

এই দুইটাকে ইমপ্লিমেন্ট করে একটা উদাহরণ দেখা যাক।

class MyClass:

"""A custom class for nothing"""

def \_\_init\_\_(self, var):

self.var = var

def \_\_del\_\_(self):

del self.var

#### কমপারিজন ম্যাজিক মেথড

| **মেথড** | **বর্ণনা** |
| --- | --- |
| \_\_cmp\_\_(self, other) | সমস্ত কমপারিজন অপারেটরের বাপ। |
| \_\_eq\_\_(self, other) | == অপারেটরের কাজ করে। |
| \_\_ne\_\_(self, other) | != অপারেটরের কাজ করে। |
| \_\_lt\_\_(self, other) | < অপারেটরের কাজ করে। |
| \_\_gt\_\_(self, other) | > অপারেটরের কাজ করে। |
| \_\_le\_\_(self, other) | <= অপারেটরের কাজ করে। |
| \_\_ge\_\_(self, other) | >= অপারেটরের কাজ করে। |

কমপারিজন ম্যাজিক মেথডের একটা ইমপ্লিমেন্টেশন দেখা যাক।

class MyClass:

"""A custom class for nothing"""

def \_\_init\_\_(self, var):

self.var = var

def \_\_del\_\_(self):

del self.var

def \_\_gt\_\_(self, other):

return len(other) > len(self.var)

def \_\_lt\_\_(self, other):

return len(other) < len(self.var)

def \_\_ge\_\_(self, other):

return len(other) >= len(self.var)

def \_\_le\_\_(self, other):

return len(other) <= len(self.var)

#### অ্যারিথমেটিক ম্যাজিক মেথড

| **মেথড** | **বর্ণনা** |
| --- | --- |
| \_\_add\_\_(self, other) | + অপারেটরের কাজ করে। |
| \_\_sub\_\_(self, other) | - অপারেটরের কাজ করে। |
| \_\_mul\_\_(self, other) | \* অপারেটরের কাজ করে। |
| \_\_div\_\_(self, other) | / অপারেটরের কাজ করে। |
| \_\_mod\_\_(self, other) | % অপারেটরের কাজ করে। |
| \_\_pow\_\_ | \*\* অপারেটরের কাজ করে। |
| \_\_floordiv\_\_(self, other) | // অপারেটরের কাজ করে। |

অ্যারিথমেটিক ম্যাজিক মেথডের একটা ইমপ্লিমেন্টেশন দেখা যাক।

>>> a = 5

>>> b = 9

>>> a.\_\_add\_\_(b)

14

>>> b.\_\_sub\_\_(a)

4

>>> a.\_\_mul\_\_(b)

45

>>> b.\_\_mul\_\_(a)

45

>>> b.\_\_mod\_\_(a)

4

>>> b.\_\_pow\_\_(a)

59049

>>> b.\_\_floordiv\_\_(a)

1

>>

#### টাইপ কনভার্সন ম্যাজিক মেথড

| **মেথড** | **বর্ণনা** |
| --- | --- |
| \_\_int\_\_(self) | ইন্টিজারে কনভার্ট করে। |
| \_\_float\_\_(self) | ফ্লোটে কনভার্ট করে। |
| \_\_str\_\_(self) | হিউম্যান-রিড্যাবল স্ট্রিংয়ে কনভার্ট করে। |
| \_\_repr\_\_(self) | মেশিন-রিড্যাবল স্ট্রিংয়ে কনভার্ট করে। |

টাইপ কনভার্সন ম্যাজিক মেথডের একটা ইমপ্লিমেন্টেশন দেখা যাক।

>>> a = 5

>>> type(a)

<class 'int'>

>>> a = a.\_\_float\_\_()

>>> type(a)

<class 'float'>

>>> a = a.\_\_str\_\_()

>>> type(a)

<class 'str'>

>>> a = a.\_\_repr\_\_()

>>> type(a)

<class 'str'>

#### আরো কিছু ম্যাজিক মেথড

| **মেথড** | **বর্ণনা** |
| --- | --- |
| \_\_len\_\_(self) | কন্টেইনারের লেংথ রিটার্ন করে। |
| \_\_iter\_\_(self) | কন্টেইনারের জন্য ইটারেটর রিটার্ন করে। |
| \_\_contains\_\_(self, item) | আইটেমটি কন্টেইনারের মেম্বার কিনা টেস্ট করে। |

এদের কিছু ইমপ্লিমেন্টেশন দেখা যাক।

এদের কিছু ইমপ্লিমেন্টেশন দেখা যাক :

>>> a = 'Bangladesh'

>>> b = 'desh'

>>> a.\_\_len\_\_()

10

>>> a.\_\_iter\_\_()

<str\_iterator object at 0x7f4c2c3c5a58>

>>> list(a.\_\_iter\_\_())

['B', 'a', 'n', 'g', 'l', 'a', 'd', 'e', 's', 'h']

>>> a.\_\_contains\_\_(b)

True

সামনের চাপ্টারগুলোতে আমরা হয়ত এদের আরো কিছু ইমপ্লিমেন্টেশন দেখব। তার আগ পর্যন্ত ম্যাজিক মেথডের ম্যাজিক শো এখানেই স্থগিত।

## পলিমর্ফিজম

মাঝে মাঝে আমাদের কিছু প্রোগ্রাম লিখতে হয়, যাদের সবার আকৃতি একই রকম - কিন্তু মানুষের মত তারা সবাই আলাদা আলাদা। সেরকম ক্ষেত্রে আমরা একটা সাধারণ প্রোগ্রাম লিখে নেই, যেটা ওদের মূল আকৃতিটুকু ধরে রাখবে। তারপর বাকিদেরকে লিখি সেই মূল আকৃতির উপর ভিত্তি করে। এই কনসেপ্টটাকে বলে পলিমরফিজম (polymorphism)।

একটা ছোট্ট সহজ উদাহরণ দেই। ধরো আমার বাসায় আমি একটা ছোট্ট চিড়িয়াখানা বানাবো হুকুশ পাকুশের জন্য। আমার একটা বাঘ থাকবে, আর বাঘটা বলবে "হালুম!"। আর থাকবে বিড়াল, ইঁদুর আর একটা ছোট্ট বাচ্চা কুকুর। বিড়ালটা বলবে "মিয়াও!", ইঁদুরটা বলবে "কিঁচ কিঁচ" আর বাচ্চা কুকুরটা বলবে, "হুফ! হুফ!"। আর ওদের সবার একটা করে নাম থাকবে। তো দেখো - ওদের মূল আকৃতিটা হচ্ছে - ওদের প্রত্যেকের নাম আছে আর সবার একটা ডাকও আছে।

তাহলে আমাদের প্রোগ্রামটা হবে এরকম

class Pet:

**def** \_\_init\_\_(*self*, **name**):

*self*.name **=** name

**def** talk(*self*):

**pass**

class Cat(*Pet*):

**def** talk(*self*):

        print(*self*.name **+** ": meow!")

class Dog(*Pet*):

**def** talk(*self*):

        print(*self*.name **+** ": hoof! hoof!")

class Tiger(*Pet*):

**def** talk(*self*):

        print(*self*.name **+** ": halum!")

class Mouse(*Pet*):

**def** talk(*self*):

        print(*self*.name **+** ": kich! kich!")

cat **=** Cat("bilai")

dog **=** Dog("kuttush")

tiger **=** Tiger("bug")

mouse **=** Mouse("idur")

cat.talk()

dog.talk()

tiger.talk()

mouse.talk()

এখানে \_\_init\_\_ ফাংশনটা হচ্ছে আমাদের আগে একটা set ফাংশন ছিলো না? ঠিক ওরকম। পার্থক্য হচ্ছে, এভাবে লিখলে, ইনস্ট্যান্স তৈরী করার সময়ই সেট করে নেয়া যায়। যেমন দেখো আমরা এখানে অবজেক্টটা তৈরী করার সময়ই ব্র্যাকেটের ভিতরেই নামটা বলে দিচ্ছি।

Pet হচ্ছে আমাদের মূল ক্লাস, যেটার হচ্ছে বাকি সব ক্লাসগুলোর মূল আকৃতি। আমরা এই ধরণের ক্লাসকে বলি অ্যাবস্ট্রাক্ট ক্লাস। বাকি সব ক্লাসগুলোতে আমরা করছি Pet কিভাবে কথা বলবে, সেটা ঠিক করে দিচ্ছি। এইটাই হচ্ছে পলিমর্ফিজম।

পলিমরফিজম কোডকে তুলনামূলকভাবে সহজ আর গোছানো করে দেয়। যখন প্রোগ্রামাররা একটা বড় সিস্টেম তৈরী করে, তখন ডিজাইনটা খুব গুরুত্বপূর্ণ হয়ে যায়। ডিজাইন যদি হিজিবিজি হিজিবিজি হয় তখন কোড করতে, প্রোগ্রামটাকে ঠিকঠাক মতো রান করতে খবর হয়ে যায়। আর যখন ডিজাইন খুব সহজ, সুন্দর আর গুটুশগাটুশ টাইপের হয় তখন কোডগুলোও সহজ আর সুন্দর হয় - তখন কোডটাকে গড়গড় করে কবিতার মতো করে পড়ে ফেলা যায়। আর মুগ্ধ হয়ে তাকিয়ে থাকা যায় অনেক অনেক সময় ধরে।

## ডাটা হাইডিং/encapsulation

অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং -এ এনক্যাপসুলেশন (Encapsulation) একটি গুরুত্বপূর্ণ কনসেপ্ট যার অর্থ কিছু ভ্যারিয়েবল এবং ফাংশনকে একত্রিত করে একটি সিঙ্গেল ইউনিট হিসেবে প্রকাশ করা। এই কনসেপ্টে একটি ক্লাসের ভ্যারিয়েবল গুলোকে অন্য ক্লাস এর কাছ থেকে আড়ালে রাখা হয় এবং শুধুমাত্র ওই ক্লাসের নির্দিষ্ট মেথডের মাধ্যমে অ্যাক্সেস করার পারমিশন থাকে। এজন্য এই কনসেপ্টকে **ডাটা হাইডিং** -ও বলা হয়ে থাকে। অন্যভাবে বলা হয়, একটি ক্লাসের ইমপ্লিমেন্টেশন ডিটেইল গুলো আড়ালে রাখা।

সাধারণ অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং এর চারটি গুরুত্বপূর্ণ কনসেপ্ট হচ্ছে - **encapsulation, inheritance, polymorphism, and abstraction.**

অন্যান্য প্রোগ্রামিং ল্যাঙ্গুয়েজে একটি ক্লাসের অ্যাট্রিবিউট ও মেথড গুলোকে নির্দিষ্ট কিওয়ার্ড (অ্যাক্সেস মডিফায়ার) ব্যবহার করে প্রাইভেট বা প্রটেক্টেড হিসেবে ডিফাইন করে এই উদ্দেশ্য পূরণ করা হয়ে থাকে। এর মাধ্যমে ওই ক্লাসের ওই নির্দিষ্ট মেথড বা অ্যাট্রিবিউট গুলোকে বাইরের ক্লাস থেকে অ্যাক্সেস করা থেকে বিরত রাখা হয়।

**কিন্তু,** পাইথনে এই বিষয়টাকে একটু আলাদাভাবে দেখা হয়। বলা হয়ে থাকে - "we are all consenting adults here" অর্থাৎ - একটি ক্লাসের কোন এলিমেন্টকে শক্তভাবে বাইরের অ্যাক্সেস থেকে বিরত রাখার ব্যবস্থা করা উচিৎ নয়। আর তাই, পাইথনে সত্যিকারের কোন পদ্ধতি নেই যার মাধ্যমে একটি ক্লাসের অ্যাট্রিবিউট বা মেথডকে প্রাইভেট হিসেবে ডিফাইন করা যেতে পারে। বরং, এধরনের এলিমেন্ট গুলোকে বাইরে থেকে অ্যাক্সেস করতে নিরুৎসাহিত করা হয় এবং এগুলো যে আসলে ক্লাসের ইমপ্লিমেন্টেশন ডিটেইল তা প্রকাশ করার মাধ্যমে এগুলোর সরাসরি অ্যাক্সেস/ব্যবহার বন্ধ রাখতে বলা হয়।

**উইকলি প্রাইভেট** অ্যাট্রিবিউট এবং মেথডের নামের শুরুতে একটি আন্ডারস্কোর ব্যবহার করে এরকম প্রাইভেট এলিমেন্ট গুলোকে ডিফাইন করা হয়ে থাকে। আবার বলতে হচ্ছে - এভাবে প্রাইভেট এলিমেন্ট হিসেবে ডিফাইন করে এটাই প্রকাশ করা হয় যে, বাইরের কোড থেকে এগুলো অ্যাক্সেস করার দরকার নেই বা উচিৎ নয়। কিন্তু তার মানে এই না যে এগুলোকে অ্যাক্সেস করা যাবে না।

class Queue:

**def** \_\_init\_\_(*self*, **contents**):

*self*.\_hiddenlist **=** list(contents)

**def** push(*self*, **value**):

*self*.\_hiddenlist.insert(0, value)

**def** pop(*self*):

**return** *self*.\_hiddenlist.pop(**-**1)

**def** \_show\_list(*self*):

**return** *self*.\_hiddenlist

queue **=** Queue([1, 2, 3])

print(queue.\_hiddenlist)

queue.push(0)

print(queue.\_hiddenlist)

queue.pop()

print(queue.\_hiddenlist)

print(queue.\_show\_list())

উপরের উদাহরণে, কিছু উইকলি প্রাইভেট এলিমেন্ট ডিফাইন করা থাকলেও সেগুলো ক্লাসের বাইরে থেকে অ্যাক্সেস করা গেছে।

**কিন্তু হ্যাঁ**, এভাবে ডিফাইন করা ভ্যারিয়েবল নিয়ে তৈরি একটি পাইথন ফাইলকে মডিউল হিসেবে ইম্পোরট করলে ওই প্রাইভেট ভ্যারিয়েবল গুলো ইম্পোরট হয় না। এতে করে এগুলোর সরাসরি অ্যাক্সেস বাধাগ্রস্ত রাখা হয়। অর্থাৎ, from module\_name import \* কোড ব্যবহার করলেও module\_name এর মধ্যে থাকা আন্ডারস্কোর ওয়ালা ভ্যারিয়েবল গুলো ইম্পোরট হবে না।

উদাহরণ,

# myfile.py

\_my\_private\_variable = 10

# data-hiding-test.py

from myfile import \*

print(\_my\_private\_variable)

data-hiding-test.py ফাইলকে রান করলে আউটপুট আসবে,

Traceback (most recent call last):

File "/Users/nuhil/Documents/Python/data-hiding-test.py", line 4, in <module>

print(\_my\_private\_variable)

NameError: name '\_my\_private\_variable' is not defined

**ট্রংলি প্রাইভেট** এ ধরনের অ্যাট্রিবিউট ও মেথডের নামের শুরুতে ডাবল আন্ডারস্কোর ব্যবহার করা হয়। পাইথন এরকম নামের অ্যাট্রিবিউট বা মেথড পেলে এগুলোর নামকে আরেক্টু পরিবর্তন করে ফেলে। এতে করে ক্লাসের বাইরে থেকে সেই ডিফাইন করা নামে এদেরকে আর অ্যাক্সেস করা যায় না।

মূলত অ্যাক্সেস রোধ করার জন্য এরকম করা হয় না বরং একটি ক্লাসের সাবক্লাসে যদি একই নামের এলিমেন্ট থাকে তাহলে যেন সেগুলোর সাথে কনফ্লিক্ট না করে।

উদাহরণ,

class Spam:

\_\_egg = 7

def print\_egg(self):

print(self.\_\_egg)

s = Spam()

s.print\_egg()

print(s.\_\_egg)

7

Traceback (most recent call last):

File "/Users/nuhil/Documents/Python/myfile.py", line 10, in <module>

print(s.\_\_egg)

AttributeError: 'Spam' object has no attribute '\_\_egg'

যদিও নিচের মত করে ঠিকি ওই প্রাইভেট এলিমেন্টকে অ্যাক্সেস করা যা

class Spam:

\_\_egg = 7

def print\_egg(self):

print(self.\_\_egg)

s = Spam()

s.print\_egg()

print(s.\_Spam\_\_egg)

অর্থাৎ, পাইথন আসলে আড়ালে, ডাবল আন্ডারস্কোর ওয়ালা এলিমেন্টের নামের সাথে তার ক্লাসের নামটি জুড়ে দেয় আর তাই s.\_Spam\_\_egg ব্যবহার করে Spam ক্লাসের \_\_egg কে অ্যাক্সেস করা হয়েছে।

## স্ক্লাস মেথড ও ট্যাটিক মেথড

**ক্লাস মেথড** আমরা আগেই জেনেছি, ইন্সট্যান্স মেথডকে একটি ক্লাসের ইন্সট্যান্স এর মাধ্যমে কল করা হয় এবং সেই ইন্সট্যান্সকে ওই মেথডের self প্যারামিটার হিসেবে পাঠানো হয় (ক্লাসের মেথড গুলোর প্রথম প্যারামিটার হিসেবে self ডিফাইন করতে হয়)।

কিন্তু ক্লাস মেথড একটু আলাদা। এ ধরনের মেথডকে সরাসরি ক্লাসের মাধ্যমেই কল করা হয় এবং সেই ক্লাস কে ওই মেথডের cls প্যারামিটার হিসেবে পাঠানো হয় (ক্লাস মেথডের প্রথম প্যারামিটার সাধারণত cls হয়ে থাকে)।

classmethod ডেকোরেটর ব্যবহার করে ক্লাস মেথড নির্দেশিত করা হয়। যেমন,

class Rectangle:

def \_\_init\_\_(self, width, height):

self.width = width

self.height = height

​

def calculate\_area(self):

return self.width \* self.height

​

@classmethod

def new\_square(cls, side\_length):

return cls(side\_length, side\_length)

​

square = Rectangle.new\_square(5)

print(square.calculate\_area())

উপরের উদাহরণে, new\_square একটি ক্লাস মেথড। আর তাই একে আমরা ক্লাসের মাধ্যমেই কল করতে পারি। একটি বিষয় লক্ষণীয় যে, এ ধরনের মেথডের প্রথম প্যারামিটার হিসেবে cls তথা সেই ক্লাসকেই পাঠানো হয়।

ইন্সট্যান্স মেথডের self এবং ক্লাস মেথডের cls এর নামকরণ শুধুই একটু কনভেনশন। আলাদা নামও চাইলে ব্যবহার করা যেতে পারে।

আর সেই new\_square মেথডের প্যারামিটার হচ্ছে একটি। তার মানে আমরা এই মেথডকে কল করতে পারছি একটি মাত্র প্যারামিটার দিয়েই এবং যেহেতু তার প্রথম প্যারামিটার হিসেবে সেই ক্লাসটি নিজেই নির্দেশিত হচ্ছে তার মানে ওই new\_square মেথডের মধ্যে থেকে আমরা সেই ক্লাস তথা Rectangle কেই ধরে সেটাকে ইন্সট্যান্সিয়েট করতে পারি। return cls(side\_length, side\_length) লাইনে আমরা ঠিক সেই কাজটিই করছি অর্থাৎ, Rectangle ক্লাসের কন্সট্রাক্টর এর দুটি প্যারামিটারের চাহিদা মোতাবেক দুটি প্যারামিটারই পাঠিয়ে ফ্রেশ একটি Rectangle ক্লাসের অবজেক্ট ইনিশিয়েট করেছি এবং রিটার্ন করছি।

তার মানে, square = Rectangle.new\_square(5) লাইনের মাধ্যমে আমরা square ভ্যারিয়েবলের মধ্যে বস্তুত স্ট্যান্ডার্ড Rectangle ক্লাসের অবজেক্ট পাচ্ছি। আর তাই শেষ লাইনে সেই অবজেক্টের মেথড তথা একটি স্বাভাবিক ইন্সট্যান্স মেথড calculate\_area কে কল করে আশানরুপ ফল পাই।

উপরের প্রোগ্রামের আউটপুট,

25

ক্লাস মেথডের বহুল ব্যবহার হতে পারে ফ্যাক্টরি মেথড তৈরি জন্য যেখানে একটি ক্লাসের অবজেক্ট দরকার হলে আমরা চাইলে ওই ক্লাসের কন্সট্রাক্টরের চাহিদা মোতাবেক আর্গুমেন্ট না পাঠিয়েও আরেকটি মেথডের মাধ্যমে (এ ক্ষেত্রে ক্লাস মেথড) ওই ক্লাসের স্বাভাবিক একটি অবজেক্ট পেতে পারি।

**স্ট্যাটিক মেথড** স্ট্যাটিক মেথড অনেকটাই ক্লাস মেথডের মত যেমন, সরাসরি ক্লাস এর মাধ্যমেই কল করা যায়। কিন্তু আবার একটু আলাদা যেমন, ক্লাস মেথডের মত এই মেথড এর প্রথম প্যারামিটার হিসেবে কলার ক্লাসকে পাঠাতে হয় না। আর তাই, সহজ ভাবে স্ট্যাটিক মেথডকে নরমাল ফাংশনের সাথে তুলনা করা হয় কিন্তু যা বিশেষত ক্লাসের এলিমেন্ট অর্থাৎ ক্লাস বা ক্লাসের ইন্সট্যান্স এর মাধ্যমে কল করা যায়। staticmethod ডেকোরেটর ব্যবহার করে স্ট্যাটিক মেথড ডিফাইন করা হয়।

উদাহরণ,

class Pizza:

def \_\_init\_\_(self, toppings):

self.toppings = toppings

​

@staticmethod

def validate\_topping(topping):

if topping == "pineapple":

raise ValueError("No pineapples!")

else:

return True

​

ingredients = ["cheese", "onions", "spam"]

if all(Pizza.validate\_topping(i) for i in ingredients):

pizza = Pizza(ingredients)

উপরের প্রোগ্রামটি কোন এক্সেপশন ছাড়াই রান করবে। এখানে validate\_topping একটি স্ট্যাটিক মেথড। ফর লুপ ব্যবহার করে Pizza.validate\_topping(i) স্টেটমেন্টের মাধ্যমে ingredients লিস্টের প্রত্যেকটি এলিমেন্টের জন্য আমরা স্ট্যাটিক মেথডটিকে কল করে একটা সাধারণ চেকিং এর কাজ সম্পন্ন করেছি এবং তা সফল হলে Pizza ক্লাসের অবজেক্ট তৈরি করেছি।

# Regular expression

পাইথনে রেগুলার এক্সপ্রেশন নিয়ে কাজ করার জন্য বিল্ট ইন মডিউল হিসেবে আছে re নামের মডিউল। রেগুলার এক্সপ্রেশন ব্যবহারের সময় প্যাটার্ন খোঁজার জন্য যে স্পেশাল এক্সপ্রেশন বা সহজ করে বলতে সার্চ টার্ম ডিফাইন করতে হয় সেটা সাধারণত r"expression" এভাবে ডিফাইন করতে হয়। r দিয়ে Raw স্ট্রিং বোঝানো হয়।

অর্থাৎ এর মধ্যে কোন রকম ক্যারেক্টার এক্সকেইপ করা হয় না যা রেগুলার এক্সপ্রেশনের ব্যবহারকে আরও ইফসিয়েন্ট করে তোলে।

**import** re

cool**=**r'mah'

lol**=**re.match(cool,'mahmud')

**if** lol:

    print('match')

**else**:

    print('not')

প্রথমেই re মডিউলকে import করে নেয়া হয়েছে যাতে করে এর মধ্যেকার সব ফাংশনকে সহজে আমাদের প্রোগ্রামে ব্যবহার করতে পারি। এরপর একটি ভ্যারিয়েবলে Raw ফরম্যাটে Bangla স্ট্রিং টিকে স্টোর করা হয়েছে। বস্তুত এই প্যাটার্নটিকেই আমরা একটু পরে আরেকটি স্ট্রিং এর মধ্যে খুঁজব। এরপর re এর match ফাংশন কে কল করা হয়েছে এবং এর দুটো আর্গুমেন্ট পাঠিয়ে দেয়া হয়েছে - একটি হচ্ছে কি ম্যাচ করে দেখতে চাই, আরেকটি হচ্ছে কোথায় ম্যাচ করে দেখতে চাই। match ফাংশন একটি স্ট্রিং এর শুরুতে ডিফাইন করা প্যাটার্নকে খুঁজে দেখে।

এই অপারেশনটির রেজাল্ট ষ্টোর করা হয়েছে result ভ্যারিয়েবলে। সাধারণত, নির্দিষ্ট প্যাটার্ন ম্যাচ পাওয়া গেলে এখানে একটি ম্যাচ সম্বলিত অবজেক্ট পাওয়া যাবে আর ম্যাচ পাওয়া না গেলে None রিটার্ন আসবে। এরপরের if-else এর কাজ টুকু সবাই বুঝতে পারছেন আশা করি।

Example:

#user keyboard input

**import** re

**def** fun():

    cool**=**r'mah'

    sos**=**input('enter a string :')

**if** re.match(cool,sos):

        print('match')

**else**:

        print('not')

fun()

এরকম আরও মজার সব ফাংশন আছে re মডিউলে। যেমন - search, findall, finditer ইত্যাদি। আমরা নিচে একটি প্রোগ্রামের মধ্যেই সব গুলোর ব্যবহার দেখবো এবং তারপর বিশ্লেষণ করবো।

Example:

import re

pattern = r"Bangladesh"

if re.search(pattern, "There is country named Bangladesh in south asia!"):

print("Match Found!")

else:

print("No match")

pattern = r"bangla"

print(re.findall(pattern, "Bangladeshi bangla and indian bangla are differnet."))

example:

**import** re

**def** fun():

    cool**=**r'mah'

    sos**=**input('enter a string :')

**if** re.search(cool,sos):

        print('match')

**else**:

        print('not')

fun()

example :

#using class

**import** re

class one:

**def** fun1(*self*):

        lol**=**r'mahmud'

        sos**=**input('enter a string name :')

**if** re.match(lol,sos):

            print('yes')

**else**:

            print('not')

class two(*one*):

**def** fun2(*self*):

        lol**=**r'i am mahmud'

        sos**=**input('enter a paragrap: ')

**if** re.search(lol,sos):

            print('yes found')

**else**:

            print('not found')

a**=**two()

a.fun1()

a.fun2()

example:

#init method with super

**import** re

class one:

**def** \_\_init\_\_(*self*):

        lol**=**r'mahmud'

        sos**=**input('enter a string name :')

**if** re.match(lol,sos):

            print('yes')

**else**:

            print('not')

class two(*one*):

**def** \_\_init\_\_(*self*):

        super().\_\_init\_\_()

        lol**=**r'i am mahmud'

        sos**=**input('enter a paragrap: ')

**if** re.search(lol,sos):

            print('yes found')

**else**:

            print('not found')

a**=**two()

example :

all re method with class inheritance e

**import** re

class one:

**def** fun1(*self*):

        a**=**'mahmud'

        b**=**input('Enter a string name:')

**if** re.match(a,b):

            print('matched')

**else**:

            print('not matched')

class two():

**def** fun2(*self*):

        a**=**'i am mahmud hossain'

        b**=**input('Enter  a paragrap: ')

**if** re.search(a,b):

            print('found searched')

**else**:

            print('not founs search')

class three(*one*,*two*):

**def** fun3(*self*):

        a**=**'mahmud'

        b**=**input('enter a list name:')

        c**=**re.findall(a,b)

        print(c)

lol**=**three()

lol.fun1()

lol.fun2()

lol.fun3()

example :

#all re method, by using init class, and super

**import** re

class one:

**def** \_\_init\_\_(*self*):

        a**=**'mahmud'

        b**=**input('Enter a string name:')

**if** re.match(a,b):

            print('matched')

**else**:

            print('not matched')

class two():

**def** \_\_init\_\_(*self*):

        a**=**'i am mahmud hossain'

        b**=**input('Enter  a paragrap: ')

**if** re.search(a,b):

            print('found searched')

**else**:

            print('not founs search')

class three(*one*):

**def** \_\_init\_\_(*self*):

        super().\_\_init\_\_()

*#two().\_\_init\_\_()*

        a**=**'mahmud'

        b**=**input('enter a list name:')

        c**=**re.findall(a,b)

        print(c)

lol**=**three()

search ফাংশন এর মাধ্যমে একটি প্যাটার্নকে একটি স্ট্রিং এর যেকোনো যায়গায় খুঁজে দেখা হয়। match এর মত শুধু শুরুতে চেক করার মত নয়। findall ফাংশনও search এর মত সব যায়গায় ম্যাচ খুঁজে দেখে এবং খুঁজে পাওয়া সব গুলো ম্যাচকে একটি লিস্ট হিসেবে রিটার্ন করে। উপরের প্রোগ্রামে এই দুটি ফাংশনের ব্যবহারকেই দেখানো হয়েছে।

### রিটার্ন অবজেক্টের কিছু মেথড

### Regex library link

<https://docs.python.org/3/library/re.html>

আগেও একবার বলা হয়েছে যে - রেগুলার এক্সপ্রেশন সম্বলিত একটি স্টেটমেন্ট বা অপারেশন একটি অবজেক্ট রিটার্ন করে। এই রিটার্ন করা অবজেক্টের আবার অনেক গুলো সুবিধাজনক মেথড থাকে যেগুলো ব্যবহার করে আরও কিছু গুরুত্বপূর্ণ কাজ সম্পন্ন করা যায়। যেমন - group, start, end, span ইত্যাদি।

উদাহরণ,

**import** re

pattern **=** r"mah"

match **=** re.search(pattern, "i am mahmud ")

**if** match:

    print(match.group())

    print(match.start())

    print(match.end())

    print(match.span())

group মেথড রিটার্ন করছে ম্যাচ হয়ে যাওয়া সাব স্ট্রিং টিকে। start, end দিয়ে স্ট্রিং এর মধ্যে হওয়া ম্যাচ এর শুরু আর শেষের ইনডেক্স বা অবস্থান জানা যায়। span এর মাধ্যমে এই ইনডেক্স দুটিকে একটি টাপল হিসেবে রিটার্ন পাওয়া যায়।

### মেটা ক্যারেক্টার

আমরা এর আগে জেনেছি যে রেগুলার এক্সপ্রেশন হচ্ছে এক ধরণের ডোমেইন স্পেসিফিক ল্যাঙ্গুয়েজ। উদাহরণ হিসেবে জেনেছি SQL এর কথা। তাই স্বাভাবিক ভাবেই এর মাধ্যমে জটিল কিছু লজিক বা প্যাটার্ন লিখতে হতেই পারে। আর তাই, রেগুলার এক্সপ্রেশন লেখার সময় কিছু মেটা ক্যারেক্টার এর ব্যবহার করতে হয়। এগুলোর ব্যবহারের মাধ্যমে বস্তুত রেগুলার এক্সপ্রেশনকে আরও ডাইনামিক ভাবে ব্যবহার করা যায়। নিচের কিছু উদাহরণ দেখলেই বিষয়টি পরিষ্কার হয়ে যাবে।

#### . (dot)

এই মেটা ক্যারেক্টারের মাধ্যমে যেকোনো ক্যারেক্টার ম্যাচ করার নির্দেশ দেয়া হয় (শুধু নিউ লাইন ক্যারেক্টার বাদে)।

**import** re

class one:

**def** fun1(*self*):

        a**=**r'mah.d'

        b**=**input('Enter a name:')

**if** re.match(a,b):

            print('yes')

**else**:

            print('no')

x**=**one()

x.fun1()

*#give mahmud, before d ,only one latter can put not more, altho it not work*

y**=**one()

y.fun1()

*#give mahmudss, after d , not latter add,altho it show false*

z**=**one()

z.fun1()

*#give mahud, its better pattent match*

Example:

import re

pattern = r"gr.y"

if re.match(pattern, "grey"):

print("Match 1")

if re.match(pattern, "gray"):

print("Match 2")

if re.match(pattern, "blue"):

print("Match 3")

উপরে আমরা একটি রেগুলার এক্সপ্রেশন ডিফাইন করেছি r"gr.y" এর মাধ্যমে। এখানে . দিয়ে ওই অবস্থানে যেকোনো ক্যারেক্টার এর সাথে ম্যাচ দেখতে বলা হয়েছে। আর তাই যখন grey বা gray এর সাথে ম্যাচ করা হয়েছে তখন রেজাল্ট সত্য এসেছে এবং একটি প্রিন্ট স্টেটমেন্ট এক্সিকিউট হয়েছে। blue এর ক্ষেত্রে তা হয় নি।

ইতোমধ্যে হয়তো খেয়াল করেছেন এক্সপ্রেশন এর শুরুতে r এর ব্যবহার। এর মাধ্যমে একটি স্ট্রিং কে Raw বা শুধুই সাধারণ স্ট্রিং হিসেবে ডিফাইন করা হয়। এতে করে রেগুলার এক্সপ্রেশন এর মধ্যে থাকা "মেটা ক্যারেক্টার" এবং ওই "মেটা ক্যারেক্টারের মতই অন্য সাধারণ ম্যাচ করার ক্যারেক্টার" এর মধ্যে পার্থক্য তৈরি করা হয়।

#### ^ এবং $

আরও দুটি বহুল ব্যবহৃত মেটা ক্যারেক্টার হচ্ছে ^ এবং $. এ দুটোর মাধ্যমে যথাক্রমে কোন একটি স্ট্রিং এর শুরু এবং শেষ চেক করে দেখা হয়। যেমন,

import re

pattern = r"^wr.te$"

if re.match(pattern, "write"):

print("Match 1")

if re.match(pattern, "wrote"):

print("Match 2")

if re.match(pattern, "writer"):

print("Match 3")

উপরের প্রোগ্রামে r"^wr.te$" এর মাধ্যমে একটি স্ট্রিং যার শুরু এবং শেষ নির্দিষ্ট অর্থাৎ যথাক্রমে w এবং e কিন্তু wr এর পর যেকোনো ক্যারেক্টার থাকতে পারে এবং সেটির পর আবার te থাকতে হবে। তাই write এবং wrote ম্যাচ করেছে।

Example:

**import** re

class one:

**def** fun1(*self*):

        a**=**r'^ma.de$' *#its means ,kono word er purbe ma takbey and ses a de takbe and ma-de maje ekta j kono later boste pare*

*#jemon masde right but massde not right*

        b**=**input('Enter a name:')

**if** re.match(a,b):

            print('yes')

**else**:

            print('no')

x**=**one()

x.fun1()

y**=**one()

y.fun1()

z**=**one()

z.fun1()

### ক্যারেক্টার ক্লাস

ক্যারেক্টার ক্লাস হচ্ছে কিছু ক্যারেক্টারের সমষ্টি বা সেট। এর মাধ্যমে এই সেটের মধ্যেকার যেকোনো একটি ক্যারেক্টারের সাথে নির্দিষ্ট কোন স্ট্রিং -কে ম্যাচ করে দেখা যায়। আবার একটি নির্দিষ্ট রেঞ্জ পর্যন্ত ক্যারেক্টার এর সাথেও ম্যাচ করা যায়। [] বন্ধনী ব্যবহার করে এবং এর মধ্যে নির্দিষ্ট কিছু ক্যারেক্টার যুক্ত করে একটি ক্যারেক্টার ক্লাস তৈরি করা হয় যা পরবর্তীতে সার্চ প্যাটার্ন হিসেবে ব্যবহার করা হয়।

**import** re

*# find any vauwel have or not*

pattern **=** r"[aeiou]"

**if** re.search(pattern, "mahmud"):

   print("vawel have ")

**else**:

   print("No vowel found!")

**if** re.search(pattern, "wswswws"):

   print("vawel have ")

**else**:

   print("No vowel found!")

### ক্যারেক্টার রেঞ্জ ম্যাচ

ক্যারেক্টার সেট তৈরি করার সময় - চিহ্ন দিয়ে একটি রেঞ্জ ডিফাইন করা হয়। যেমন, [a-z] ক্লাস দিয়ে ছোট হাতের যেকোনো ইংলিশ বর্ণ ম্যাচ করা হয়। [A-Z] দিয়ে যেকোনো বড় হাতের ইংলিশ বর্ণ। [0-9] দিয়ে যেকোনো নিউমেরিক ডিজিট ম্যাচ করে দেখা হয়, ইত্যাদি।

**import** re

pattern **=** r"[A-Z][A-Z][0-9]"

**if** re.search(pattern, "NS1 is prefix of first name server address."):

*# Found NS1 as match*

   print("OK")

**if** re.search(pattern, "You should put a second one with NS2 as prefix."):

*# Found NS2 as match*

   print("OK")

**if** re.search(pattern, "I don\'t have any nameserver."):

   print("NS3")

**else**:

   print("Not OK!")

**if** re.search(pattern, "PY3K"):

*# Found PY3 as match*

   print("OK")

অর্থাৎ উপরের [A-Z][A-Z][0-9] প্যাটার্নের মাধ্যমে একটি স্ট্রিং এর মধ্যে "দুটি বড় হাতের ইংলিশ বর্ণ এবং তার সাথেই যুক্ত একটি নিউমেরিক ডিজিট" সম্পন্ন একটি প্যাটার্ন ম্যাচ করা হচ্ছে।

Example:

**import** re

a**=**r'[A-Z][1-9][a-z]'

**if** re.search(a,'hello i am MMMMMMMMMMMMMMMMMMMMMMMMMM2ssssssssssssssssssssssssssssssssssss'):

    print('have')

**else**:

    print('not')

example:

**import** re

a**=**r'[1-5][a-o]'

b**=**'mahmud12345  mahmud12345mahmud'

**if** re.search(a,b):

    print('yes')

**else**:

    print('no')

example:

**import** re

a**=**r'[1-5][@][a-o]'

b**=**'mahmud12345  mahmud12345@mahmud'

**if** re.search(a,b):

    print('yes')

**else**:

    print('no')

### ক্যারেক্টার ক্লাসে ^ এর ব্যবহার

আমরা আগে জেনেছি যে, ^ হচ্ছে একটি মেটা ক্যারেক্টার। ক্যারেক্টার ক্লাসে ^ এর গুরুত্বপূর্ণ একটি ভূমিকা আছে। এর মাধ্যমে সাধারণ ভাবে তৈরি করা একটি ক্যারেক্টার ক্লাসের ঠিক উল্টো অর্থ ডিফাইন করা হয়। অর্থাৎ এটি একটি ক্যারেক্টার ক্লাসের অর্থকে invert করে ফেলে। অর্থাৎ [A-Z] দিয়ে যদি যেকোনো বড় হাতের ইংলিশ বর্ণের উপস্থিতি যাচাই করা হয়, তাহলে [^A-Z] দিয়ে যেকোনো বড় হাতের ইংলিশ বর্ণের অনুপস্থিতি যাচাই বা ম্যাচ করা হয়।

### গ্রুপ

রেগুলার এক্সপ্রেশনের একটি নির্দিষ্ট অংশকে বন্ধনীর মধ্যে আবদ্ধ করে একটি গ্রুপ তৈরি করা হয়।

উদাহরণ,

**import** re

pattern **=** r"egg(spam)**\***"

**if** re.match(pattern, "egg"):

    print("Match 1")

**if** re.match(pattern, "eggpamspamspameggjdddhdhdhdh"):

    print("Match 2")

**if** re.match(pattern, "spam"):

    print("Match 3")

উপড়ে (spam) একটি গ্রুপ। অর্থাৎ উপরোক্ত প্যাটার্নটি এই প্রকাশ করে যে - স্ট্রিং এর শুরুতে egg থাকবে এবং এর পর এক বা একাধিক spam ওয়ার্ড থাকবে অথবা নাও থাকতে পারে (\* দিয়ে প্রকাশ করা হয়েছে)।

গ্রুপের ম্যাচ করা কন্টেন্ট গুলকে group() ফাংশনের সাহায্যে অ্যাক্সেস করা যায়। যেমন, group() বা group(0) ব্যবহার করে পুরো ম্যাচটি অ্যাক্সেস করা যেতে পারে। নিচের মত করে,

উদাহরণ,

**import** re

pattern **=** r"a(bc)(de)(f(g)h)i"

match **=** re.match(pattern, "abcdefghijklmnop")

**if** match:

    print(match.group())

    print(match.group(0))

    print(match.group(1))

    print(match.group(2))

    print(match.groups())

### স্পেশাল গ্রুপ

অনেক রকম স্পেশাল গ্রুপের মধ্যে named group এবং non-capturing group অন্যতম।

named group এর ফরম্যাট দেখতে (?P<name>...) -এ রকম। যেখানে name হচ্ছে গ্রুপটির নাম এবং ... হচ্ছে কন্টেন্ট। এর আচরণ অন্যান্য নরমাল গ্রুপের মতই, শুধুমাত্র যেহেতু এর একটি নাম আছে তাই একে অ্যাক্সেস করার জন্য group(name) অর্থাৎ নাম ব্যবহার করা যায়। যদিও সাথে সাথে নাম্বারও ব্যবহার করা যেতে পারে। অন্যদিকে, non-capturing group এর ফরম্যাট দেখতে (?: ...) -এ রকম এবং গ্রুপ ফাংশন ব্যবহার করে একে অ্যাক্সেস করা যায় না।

**import** re

pattern **=** r"(?P<first>abc)(?:def)(ghi)"

match **=** re.match(pattern, "abcdefghi")

**if** match:

    print(match.group("first"))

    print(match.groups())

**আরও একটি মেটাক্যারেক্টার** | অর্থাৎ অথবা প্রকাশক একটি মেটা ক্যারেক্টার মাঝে মাঝে খুবি উপকারী। এটা অনেক লজিক্যাল OR অপারেটর এর মত। নিচের উদাহরণ দেখলে আরও পরিষ্কার হয়ে যাবে,

**import** re

pattern **=** r"gr(a**|**e)y"

match **=** re.match(pattern, "gray")

**if** match:

    print ("Gray is fine!")

match **=** re.match(pattern, "grey")

**if** match:

    print ("Grey is OK also!")

match **=** re.match(pattern, "griy")

**if** match:

    print ("No way, what Griy is?!!")

অর্থাৎ, প্যাটার্নটি এরকম - প্রথমে gr থাকবে, এরপর হয় a অথবা e থাকবে এবং শেষে y থাকবে। এরকম একটি ম্যাচ খুঁজবে এই প্যাটার্নটি। আর তাই উপড়ের প্রোগ্রামের আউটপুট আসবে নিচের মত,

### স্পেশাল সিকুয়েন্স

একটি ব্যাকস্ল্যাশ \ এবং সাথে একটি ক্যারেক্টার ব্যবহার করে রেগুলার এক্সপ্রেশনের জন্য একটি স্পেশাল সিকোয়েন্স তৈরি করা হয়। যেমন, একটি বহুল ব্যবহৃত সিকোয়েন্স হচ্ছে \1 বা \2 বা এরকম। এর মাধ্যমে ওই নাম্বার গ্রুপের (আগের চ্যাপ্টারে গ্রুপ নিয়ে আলোচনা আছে) এক্সপ্রেশনকে ম্যাচ করে দেখা হয়।

**import** re

pattern **=** r"(.**+**) \1"

match **=** re.match(pattern, "word word")

**if** match:

    print ("Match 1")

match **=** re.match(pattern, "?! ?!")

**if** match:

    print ("Match 2")

match **=** re.match(pattern, "abc cde")

**if** match:

    print ("Match 3")

প্রথম ম্যাচ স্টেটমেন্টটি খেয়াল করি - এখানে (.+) দিয়ে প্রথম গ্রুপে একটি খুশি মত যেকোনো স্ট্রিং চেক করা হচ্ছে আর এর পরেই \1 দিয়ে সেই গ্রুপের (group(1)) জন্য ম্যাচ হওয়া এক্সপ্রেশনকে (word) ম্যাচ করে দেখা হচ্ছে। অর্থাৎ প্রথম অংশে যা থাকবে পরের অংশেও কেবল তাই থাকলেই ম্যাচ সত্য হবে।

তৃতীয় ম্যাচ স্টেটমেন্টের ক্ষেত্রে abc এবং cde এক নয়। তাই এটি মিথ্যা হয়েছে।

\d\s **এবং** \w

\d দিয়ে ডিজিট, \s দিয়ে হোয়াইট স্পেস এবং \w দিয়ে ওয়ার্ড ক্যারেক্টার ম্যাচ করা হয়ে থাকে। ASCII মুডে এগুলো যথাক্রমে এভাবেও লেখা যায় বা একই এক্সপ্রেশন প্রকাশ করে - [0-9], [ \t\n\r\f\v], and [a-zA-Z0-9\_]

মজার বিষয় হচ্ছে এই স্পেশাল সিকোয়েন্স গুলোর বড় হাতের ভার্সন ঠিক উল্টো জিনিষ প্রকাশ করে। অর্থাৎ - \D দিয়ে সব কিছু কিন্তু ডিজিট নয় এমন ম্যাচ করে।

**import** re

pattern **=** r"(\D**+**\d)"

match **=** re.match(pattern, "Hi 999!")

**if** match:

    print("Match 1")

match **=** re.match(pattern, "1, 23, 456!")

**if** match:

    print("Match 2")

match **=** re.match(pattern, " ! $?")

**if** match:

    print("Match 3")

অর্থাৎ, (\D+\d) দিয়ে এমন একটি স্ট্রিং কে ম্যাচ করার কথা বলা হচ্ছে যার শুরুতে কিছু ক্যারেক্টার থাকবে যেগুলো আর যাই হোক ডিজিট নয়, এবং এরপরে কিছু ডিজিট থাকবে।

### আরও কিছু স্পেশাল সিকোয়েন্স

\A এবং \Z দিয়ে কোন স্ট্রিং এর শুরু এবং শেষ ম্যাচ করা হয়। \b দিয়ে ওয়ার্ডের মধ্যেকার বাউন্ডারি বা সীমা গুলোকে চিহ্নিত করা হয়। নিচের উদাহরণটি দেখলে এর ব্যবহার আরও পরিষ্কার বোঝা যাবে।

**import** re

pattern **=** r"\b(cat)\b"

match **=** re.search(pattern, "The cat sat!")

**if** match:

    print ("Match 1")

match **=** re.search(pattern, "We s>cat<tered?")

**if** match:

    print ("Match 2")

match **=** re.search(pattern, "We scattered.")

**if** match:

    print ("Match 3")

match **=** re.search(pattern, "We/cat.tered.")

**if** match:

    print ("Match 4")

match **=** re.search(pattern, "We{cat!tered.")

**if** match:

    print ("Match 5")

এখানে \b(cat)\b এর মাধ্যমে cat শব্দটিকে ম্যাচ করা হচ্ছে যাতে এর দুপাশে যেকোনো রকম ওয়ার্ড বাউন্ডারি থাকে। তাহলেই ম্যাচ সত্য হবে।

## (Gap\_\_)রেগুলার এক্সপ্রেশন

যে স্পেশাল ক্যারেক্টার-সিকুয়েন্স ব্যবহার করে এক বস্তা স্ট্রিং থেকে মহানায়ক অনন্ত জলিলের খোঁজ দ্যা সার্চকে কাজে লাগিয়ে নির্দিষ্ট কোন প্যাটার্ন বা সিকোয়েন্সের স্ট্রিং খুঁজে বের করা হয় প্রোগ্রামিংয়ের ভাষায় তাকে Regular Expression (রেগুলার এক্সপ্রেশন) বলে। অনেক সময় একে সংক্ষেপে regex বা regexp অথবা অতি সংক্ষেপে RE বলেও ডাকা হয়।

উইকিপিডিয়ার ভাষ্যমতে,

থিওরিটিক্যাল কম্পিউটার সায়েন্স ও ফরমাল ল্যাঙ্গুয়েজ থিওরি অনুযায়ী, রেগুলার এক্সপ্রেশন (র্যা শনাল এক্সপ্রেশন নামেও পরিচিত) হলো, ক্যারেক্টারের সিক্যুয়েন্স যা একটি সার্চ প্যাটার্নকে নির্দেশ করে, বিশেষ করে স্ট্রিং দিয়ে প্যাটার্ন ম্যাচিং করা অথবা স্ট্রিং ম্যাচিং করা। যেমন “find and replace” অপারেশন।

আমরা যারা সাধারণ মস্তিষ্কের মানুষজন আমাদের অবশ্য এতটা কঠিন কঠিন সংজ্ঞা জানার দরকার নাই। আমরা কাজ করতে করতে শিখব। এবার নিচের স্ট্রিংটা একটু লক্ষ্য করি।

purple alice@google.com, blah monkey bob@abc.com blah dishwasher

এইখানে তো দুইটা ইমেইল এড্রেস আছে, দুই চক্ষে সবাই দেখতে পাইতেছি। ধরেন, আমরা এই ইমেইল দুইটা পাইথন দিয়া খুঁজে বের করব। এজন্য আমরা re নামের একটা স্টান্ডার্ড মডিউল ব্যবহার করব।

import re

my\_string = "purple alice@google.com, blah monkey bob@abc.com blah dishwasher"

temp = my\_string.split(',')

for phrase in temp:

result = re.search("([\w\.-]+)@([\w\.-]+)", phrase)

print(result.group())

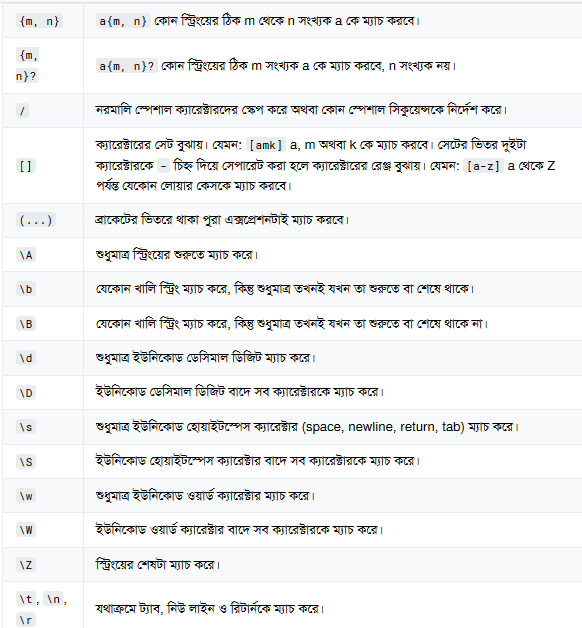
**আউটপুট**

alice@google.com

bob@abc.com

হুম, আমরা দুইটা ইমেইল খুঁজে পেয়েছি। কিন্তু কিভাবে কি করলাম? হুম, চিন্তার বিষয়। re.search("([\w\.-]+)@([\w\.-]+)", phrase) অংশটা খেয়াল করি সবাই। re.search(pattern, string, flags=0) ফাংশনের প্রথম আর্গুমেন্ট হল প্যাটার্ন (যে প্যাটার্ন ধরে আমরা খোঁজ দ্যা সার্চ চালাব), দ্বিতীয় আর্গুমেন্ট হল স্ট্রিং (যার ভিতরে খোঁজ দ্যা সার্চ চালাব), তৃতীয়টা আপাতত আমাদের মাথাব্যথার কারণ নয়। আমাদের মাথাব্যথার মূল কারণ এই প্যাটার্নটা। এটা কিভাবে লিখলাম? চলুন, সিঙ্গেল ক্যারেক্টার ম্যাচ (match) করার বেসিক প্যাটার্নগুলো শিখে নেয়া যাক।





এরকম জটিল জিনিস আমাদের ছোট্ট মস্তিষ্কে ঢোঁকার কথা না। আসলে ঢোঁকার দরকারও নাই। ব্যাপক প্রাকটিসের মাধ্যমে আমরা এগুলো আত্মস্থ করে নেব। এবার আমরা re মডিউলের কিছু ফাংশনের ব্যবহার আর উপরের সার্চ প্যাটার্নগুলোর মজার সব উদাহরণ দেখব।

### re.match(pattern, string, flags=0)

প্যাটার্ন আর স্ট্রিং - আর্গুমেন্ট দুইটার মানে তো আমরা জানিই। এবার দেখা যাক flags জিনিসটা কি। flag হল বিভিন্ন মডিফায়ার ফাংশন যারা রিজেক্সে প্রভাব-পরিবর্তন সৃষ্টি করে। এরকম কিছু flag হল re.A বা re.ASCII, re.I বা re.IGNORECASE, re.L বা re.LOCALE, re.M বা re.MULTILINE, re.S বা re.DOTALL এবং re.X বা re.VERBOSE। সামনে ধাপে ধাপে হয়ত আমরা এর সম্পর্কে অল্প-বিস্তর জানব।

এবার re.match() ফাংশনের কথায় আসি আবার। যদি রিজেক্স প্যাটার্ন স্ট্রিংয়ের শুরুতে শূন্য বা তার বেশি ম্যাচ খুঁজে পায় তাহলে এই ফাংশন একটা ম্যাচ অবজেক্ট রিটার্ন করে। আর ম্যাচ না করলে None রিটার্ন করে। আর group() বা group(num) মেথড ব্যবহার করে আমরা অবজেক্ট থেকে বিভিন্ন ঢঙে ভ্যালু পেতে পারি।

import re

my\_string = "purple alice@google.com, blah monkey bob@abc.com blah dishwasher"

temp = my\_string.split(',')

for phrase in temp:

result = re.match("([\w\.-]+)@([\w\.-]+)", phrase)

print(result)

**আউটপুট**

None

None

None রিটার্ন করেছে। এর কারণ হল match() ফাংশন শুধুমাত্র স্ট্রিংয়ের শুরুতে খোঁজ দ্যা সার্চ করে ম্যাচিং স্ট্রিং বের করে। আর search() ফাংশন পুরো স্ট্রিং জুড়েই খোঁজ দ্যা সার্চ করে ম্যাচিং স্ট্রিং বের করে। এজন্য শুরুর উদাহরণে উপযুক্ত ফলাফল পেলেও এবার পেয়েছি ঘোড়ার ডিম।

### re.sub(pattern, repl, string, count=0, flags=0)

এই ফাংশন স্ট্রিংয়ের কোন অংশ প্যাটার্নকে ম্যাচ করলে তা repl দিয়ে রিপ্লেস করে। আর প্যাটার্ন ম্যাচ না করলে কিছুই হয় না।

>>> re.sub(r'\sAND\s', ' & ', 'Baked Beans And Spam', flags=re.IGNORECASE)

'Baked Beans & Spam'

বিষয়টা মনে হয় অনেক কঠিন হয়ে গেল। তাহলে এটুকুই থাক। আমরা শুধুমাত্র ধারণা নিলাম একটা। আরেকটু বড় হয়ে আমরা বিস্তারিত ধারণা নেব। এজন্য সবাই পাইথনে অফিসিয়াল ডক ফলো করতে পারি - <https://docs.python.org/3/library/re.html> ।

# ???ডেকোরেটর

ফাংশন চাপ্টারে আমরা ফাংশন নিয়ে অল্প-বিস্তর ব্যাপক জ্ঞান লাভ করেছি। পাইথনে ফাংশন হল ফার্স্ট-ক্লাস অবজেক্ট। এর মানে হল আমরা ফাংশনকে নরমাল ভ্যালু বা ভ্যারিয়েবলের মত করে পাস করতে পারি, প্যারামিটার বা আর্গুমেন্ট হিসাবে ব্যবহার করতে পারি। বুঝলাম না মনে হয়। একটা উদাহরণ দেখা যাক।

def get\_int\_as\_str(number):

return str(number)

def print\_int(my\_function, number):

print(my\_function(number))

return

print\_int(get\_int\_as\_str, 130)

**আউটপুট**

130

এখানে আমরা print\_int() ফাংশনের ভিতর get\_int\_as\_str() ফাংশনকে আর্গুমেন্ট হিসাবে পাস করেছি। পরে সেটা দিয়ে কোডিং কারিশমা দেখাইছি। এ তো গেল ফাংশনকে আর্গুমেন্ট হিসাবে পাস করার কথা। যদি ফাংশনকেই রিটার্ন করতে চাই? এও কি সম্ভব! হুম, সব সম্ভবের পাইথনে সবই সম্ভব।

def get\_int\_as\_str(number):

print(str(number))

return

def print\_int(my\_function, number):

return my\_function(number)

print\_int(get\_int\_as\_str, 130)

**আউটপুট**

130

এখানে আমরা print\_int() ফাংশনের ভিতর get\_int\_as\_str() ফাংশনকে ও একটা ইন্টিজার ভ্যালুকে আর্গুমেন্ট হিসাবে পাস করেছি। print\_int() ফাংশনের রিটার্ন অংশে আবার get\_int\_as\_str() ফাংশনকে রিটার্ন করেছি। এইক্ষেত্রে একটা কাহিনি আছে। get\_int\_as\_str() ফাংশনটা রিটার্ন হবার সময় এক্সিকিউট হওয়া শুরু করে। তাই দিনশেষে get\_int\_as\_str() ফাংশন যা রিটার্ন করে print\_int() ফাংশনও তাই রিটার্ন করে।

আমরা কিন্তু চাইলে get\_int\_as\_str() ফাংশনটাকে print\_int() ফাংশনের ভিতরেই ডিফাইন করতে পারতাম।

def print\_int(number):

def get\_int\_as\_str(number):

print(str(number))

return

get\_int\_as\_str(number)

return

print\_int(130)

**আউটপুট**

130

উপরের প্রোগ্রামটাকে আরেকটু মডিফাই করে নিচের চেহারা দিতে পারি আমরা।

def print\_int(number):

def get\_int\_as\_str(number):

print(str(number))

return

return get\_int\_as\_str(number)

print\_int(130)

**আউটপুট**

130

নিজেদের অজান্তেই কিন্তু আমরা একটা ডেকোরেটর লিখে ফেলেছি। এইক্ষেত্রে print\_int() হল ডেকোরেটর। পাইথনের ভাষায়, ডেকোরেটর হল এমন একটা ফাংশন যা অন্য কোন ফাংশনের কার্যপরিধি কোন প্রকার মডিফিকেশন ছাড়াই এক্সটেন্ড করে। ডেকোরেটর লেখার সিনট্যাক্স হল @decorator\_name। উপরের প্রোগ্রামটাকে এখন আমরা এভাবে লিখতে পারি।

def print\_int(my\_function):

def any\_function():

return my\_function

return any\_function()

@print\_int

def get\_int\_as\_str(number):

print(str(number))

return

get\_int\_as\_str(130)

**আউটপুট**

130

বুঝলাম সবাই? ডেকোরেটর তো বুঝলাম। কিন্তু এর সুবিধা তো বোধগম্য হল না। এরজন্য আমরা আরেকটা প্রোগ্রাম দেখব। আমরা এখন একটা ডেকোরেটর ফাংশন লিখব যেটা কোন ফাংশন এক্সিকিউট হতে কত সময় নেয় তা জাতিরে জানান দিবে। এইক্ষেত্রে পাইথন স্টান্ডার্ড মডিউল time এর time() ফাংশনের সহায়তা নেব আমরা।

from time import time

def timer(any\_function):

def count\_time():

start = time()

any\_function()

stop = time()

print(stop-start, 'seconds')

return

return count\_time

@timer

def hello():

print('Hello World!')

return

@timer

def another\_function():

for item in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:

print(item)

return

hello()

another\_function()

**আউটপুট**

Hello World!

0.006817340850830078 seconds

1

2

3

4

5

6

7

8

9

10

0.041167497634887695 seconds

## বিল্ট-ইন ডেকোরেটর

কাস্টম ডেকোরেটর নিয়ে বেশ আলোচনা হয়েছে। এবার আমরা পাইথনের শক্তিশালী তিনটি বিল্ট-ইন ডেকোরেটর সম্পর্কে জানব।

### ক্লাস মেথড - @classmethod

এই ডেকোরেটরটা সম্পর্কে জানার আগে আমরা এর একটা প্রয়োগ দেখব:

class MyClass:

"""Simple Class to define something"""

def \_\_init\_\_(self):

pass

def square(self, x):

return x\*\*2

@classmethod

def cube(self, x):

return x\*\*3

if \_\_name\_\_ == "\_\_main\_\_":

myclass = MyClass()

print(myclass.square(3))

print(myclass.cube(3))

print(MyClass.cube(3))

print(MyClass.square(3))

**আউটপুট**

9

27

27

Traceback (most recent call last):

File "/home/maateen/Desktop/test.py", line 19, in <module>

print(MyClass.square(3))

TypeError: square() missing 1 required positional argument: 'x'

আমরা একটা নিরীহ গোবেচারা ধরনের ক্লাস লিখেছি। এই ক্লাসে square() এবং cube() নামের দুটি মেথডও লিখেছি। আর বিশেষায়িত দিকটা হলো cube () মেথডে ক্লাসমেথড ডেকোরেটর ব্যবহার করেছি। আউটপুটে আমরা TypeError দেখতে পাচ্ছি। square() ফাংশনের একটা পজিশনাল আর্গুমেন্ট মিসিং। কাহিনিটা ময়নাতদন্ত করা যাক।

১৫ নম্বর লাইনে আমরা MyClass-এর একটা ইনস্ট্যান্স বা অবজেক্ট তৈরি করেছি এবং ১৬ ও ১৭ নম্বর লাইনে এই অবজেক্টটা ব্যবহার করে মেথড দুটো কল করেছি। আর তারা ঠিকমতো কাজও করেছে। ১৮ নম্বর লাইনে আমরা সরাসরি ক্লাসের নাম ধরে মেথড কল করেছি। এবারও কোনো সমস্যা হয়নি। কিন্তু ১৯ নম্বর লাইনে একই কাজ করতে গিয়ে এরর দেখাল। ঘটনা কী?

ঘটনা হলো, cube() মেথডে আমরা ক্লাসমেথড ডেকোরেটর ব্যবহার করেছি কিন্তু square() মেথডে তা করিনি। যখন ক্লাসমেথড ডেকোরেটর ব্যবহার করা হয় তখন ওই ফাংশন অবজেক্ট তৈরি না করেও কল করা যায়।

### স্ট্যাটিক মেথড - @staticmethod

সবার আগে আমরা স্ট্যাটিকমেথড ডেকোরেটরের একটা প্রয়োগ দেখব :

class MyClass:

"""Simple Class to define something"""

def \_\_init\_\_(self):

pass

def square(self, x):

return x\*\*2

@staticmethod

def cube(x):

return x\*\*3

if \_\_name\_\_ == "\_\_main\_\_":

myclass = MyClass()

print(myclass.square(3))

print(myclass.cube(3))

print(MyClass.cube(3))

print(MyClass.square(3))

**আউটপুট**

9

27

27

Traceback (most recent call last):

File "test.py", line 19, in <module>

print(MyClass.square(3))

TypeError: square() missing 1 required positional argument: 'x'

ঘটনা কিন্তু আগেরটাই। আর ঘটনার কারণও সেই আগেরটাই। তাহলে আমাদের মনে দুটি প্রশ্ন উঠতে পারে। প্রথমত, cube() মেথডে প্রথম আর্গুমেন্ট হিসেবে self ব্যবহার করা হয়নি কেন? দ্বিতীয়ত, দুটি ডেকোরেটরের মধ্যে তাহলে পার্থক্য কী?

স্ট্যাটিক মেথড ডেকোরেটরের ক্ষেত্রে মেথডের প্রথম আর্গুমেন্ট ক্লাস ইনস্ট্যান্স বা self হবে না। সাধারণ ফাংশন লেখার মতো করে লিখতে হবে। ক্লাসমেথড ডেকোরেটরের ক্ষেত্রে প্রথম আর্গুমেন্ট হবে ক্লাস বা ক্লাস ইনস্ট্যান্স। দুটোর মধ্যে এটাই হলো অমিল। আর দুজনের কাজই কিন্তু এক। একটা মেথডকে সরাসরি ক্লাস থেকে কল করতে পারা যায়। এই কাজই কিন্তু আমরা ওপরে করেছি।

### প্রোপার্টি - @property

পাইথনের অস্থির কনসেপ্টগুলোর মধ্যে প্রোপার্টি একটা। এই কনসেপ্ট অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিংকে করেছে আরও পোক্ত। যাহোক, পাইথন শেলে একটা প্রোগ্রাম লিখি:

>>> class Weather:

...

... def \_\_init\_\_(self, temperature=0):

... self.temperature = temperature

...

... def to\_fahrenheit(self):

... return self.temperature \* 1.8

...

>>> weather = Weather()

>>> weather.temperature

0

>>> weather.temperature =25

>>> weather.temperature

25

>>> weather.to\_fahrenheit()

45.0

এখানে আমরা ক্লাসের ইনস্ট্যান্স থেকে এর প্রোপার্টি অ্যাক্সেস করেছি। এবার প্রোপার্টি ডেকোরেটর ব্যবহার করে আমরা একটা উদাহরণ দেখব :

class MyClass:

"""Simple Class to define something"""

def \_\_init\_\_(self, first\_name, last\_name):

self.first\_name = first\_name

self.last\_name = last\_name

@property

def full\_name(self):

return self.first\_name + ' ' + self.last\_name

if \_\_name\_\_ == "\_\_main\_\_":

myclass = MyClass('Maksudur', 'Rahman')

print(myclass.full\_name)

myclass.full\_name = 'New Name'

**আউটপুট**

Maksudur Rahman

Traceback (most recent call last):

File "test.py", line 15, in <module>

myclass.full\_name = 'New Name'

AttributeError: can't set attribute

প্রোপার্টি ডেকোরটের ব্যবহার করে আমরা full\_name() মেথডকে রিড-অনলি অ্যাট্রিবিউটে পরিণত করেছি। তাই সাধারণভাবে ডট চিহ্ন (.) দিয়ে full\_name-কে অ্যাক্সেস করতে পারছি। আর রিড-অনলি হওয়ায় এই অ্যাট্রিবিউটে নতুন কিছু দেওয়া যাচ্ছে না।

### পাইথন ডেকরেটর‍্স‍

“ডেকরেটর! ওই যে বিভিন্ন অনুষ্ঠানের সাজ সজ্জা করে মানে ডেকরেট করে….. তো পাইথনে আবার তাদের কি কাজ!”

পোস্টের টাইটেল দেখে যদি আপনার মনে এরকম প্রশ্নের উদয় হয় তাহলেও দোষের কিছু নেই। পাইথনে ডেকরেটর কিছুটা এডভান্স আর কমপ্লেক্স টপিক। তবে চিন্তা নেই, এই পোস্টে আমরা একটু সহজ ভাবে, ধাপে ধাপে জানার চেষ্টা করবো পাইথনে ডেকরেটর জিনিষটা কী, কীভাবে কাজ করে আর কীভাবেই বা ব্যাবহার করে।

ঠিক ঠাক ভাবে বললে, ডেকরেটর হল এক ধরনের callable যা অন্য callable এর ফাংশনালিটিকে মডিফাই করে। আরেকটু সহজ করে বললে, ডেকরেটর হল এক ধরনের ফাংশন যা অন্য ফাংশনের ফাংশনালিটিকে মডিফাই করে। [ডেকরেটর ক্লাসও হয়, তবে এ পোস্টে সেটা আলোচনা করবো না।] তো ধাপে ধাপে শুরু করা যাক। আমি ধরে নিচ্ছি ভ্যারিয়েবলের স্কোপ রেজ্যুলেশন সম্পর্কে আমাদের মোটামুটি ভাল ধারনা আছে, তাই এ সম্পর্কে আর বাড়তি কিছু লিখলাম না।

### ফাংশনের কারিকুরি

এই কোড ব্লক টা দেখি:

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | def hello():      print("Hello World!")      hello()  # output: Hello World!    hi = hello    # interesting!    print(hi)  # output: <function hello at 0x7ff330b412f0>    hi()  # output: Hello World! |

৮ নম্বর লাইন থেকে কোডটা ইন্টারেস্টিং হওয়া শুরু করেছে। আমরা জানি পাইথনে সব কিছুই এক একটা অবজেক্ট। ফাংশনও। এই লাইনে আমরা hello কে hi তে এসাইন করেছি। লক্ষনীয়, এখানে hello এর পাশে () (ব্র্যাকেট/প্যারেন্থেসিস) দেই নি। অর্থাৎ এখানে hello ফাংশনটি এক্সিকিউট বা কল হয় নি। ১০ নম্বর লাইনের আউটপুট দেখলে ব্যাপারটা আরো পরিষ্কার হবে। আর ১৩ নম্বর লাইনে hi কে কল করা হয়েছে, আউটপুট পেয়েছি ঠিক hello এর মত।

### ফাংশনের ভেতর ফাংশন!

হ্যা, পাইথনে আমরা ফাংশনের ভেতর ফাংশন ডিফাইন করতে পারি। অন্যভাবে বললে আমরা নেস্টেড ফাংশন বানাতে পারি। এরকম:

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | def hello():      print("Inside hello")        def nested():          print("Inside nested")        nested()      print("Inside hello again")      hello() |

আউটপুট:



|  |  |
| --- | --- |
| 1  2  3 | Inside hello  Inside nested  Inside hello again |

আরেকটি কোড ব্লক দেখি:

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | def count(value):        def increment():          return value + 1        print("Incremented value of {} is {}".format(value, increment()))      count(5)  # output: Incremented value of 5 is 6 |

শুরুতেই লিখেছিলাম ভ্যারিয়েবলের স্কোপ রেজ্যুলেশন সম্পর্কে লিখবো না। কোডটা একটু ভাল মত লক্ষ্য করলেই আশা করি বুঝতে পারবেন।

### ফাংশন থেকে ফাংশন রিটার্ন!

ফাংশন থেকে ইচ্ছা করলে আমরা ফাংশন রিটার্নও করতে পারি! এই কোডটা দেখলে ব্যাপারটা পরিষ্কার হবে:

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | def hello():        def nested():          print("Inside nested")        return nested      hi = hello()    print(hi)  # output: <function hello.<locals>.nested at 0x7f1ed4e94488>    hi()  # output: Inside nested |

৯ নম্বর লাইনে hello() কে কল করায় এটা nested কে রিটার্ন করেছে, তা এসাইন হয়েছে hi তে। পূর্বের কোড ব্লক গুলো ফলো করলে এটি সহজেই বোঝা যাবে।

আরেকটি কোড ব্লক দেখি:

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | def count(value):        def increment(n):          return value + n        return increment      counter = count(5)  print(counter(1))    # Guess the output! |

কি, আউটপুট গেস করেছেন? এটা আর এক্সপ্লেইন করবো না। আউটপুট হবে 6।

### ফাংশনের আর্গুমেন্ট/প্যারামিটার হিসেবে ফাংশন

সরাসরি একটা কোড স্নিপেট দেখে ফেলি:

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | def hello():      print("Hello World!")      def hi(func):      print("Hi!")      func()      hi(hello) |

এখানে hi ফাংশনের প্যারামিটার হিসেবে hello কে পাস করা হয়েছে। hi এর ভেতর hello কল হয়েছে। আউটপুট হবে এরকম:



|  |  |
| --- | --- |
| 1  2 | Hi!  Hello World! |

## ডেকরেটর

এখন হচ্ছে মূল বিষয়, ডেকরেটর। আমরা আগেই জেনেছি, ডেকরেটর হচ্ছে এমন ফাংশন যা অন্য ফাংশনের ফাংশনালিটি মডিফাই করে। এখন তাহলে একটু সাজানো গুছানো উদাহরণ দেখে নেই:

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | def mydecorator(func):        def wrapper():          print("Before calling func()")          func()          print("After calling func()")        return wrapper      def hello():      print("Hello World!")      hello = mydecorator(hello)  hello() |

এখানে mydecorator ফাংশনটি প্যারামিটার হিসেবে আরেকটি ফাংশন এক্সপেক্ট করছে। এর মধ্যকার wrapper ফাংশনটিতে প্যারামিটারে পাওয়া ফাংশন কল করার আগে এবং পরে কিছু কাজ হচ্ছে। আর mydecorator থেকে wrapper কে রিটার্ন করা হচ্ছে। ১৫ নম্বর লাইনে mydecorator কে hello প্যারামিটার দিয়ে কল করা হয়েছে। রিটার্ন ভ্যালু এসাইন করা হয়েছে আবার hello তে। অর্থাৎ mydecorator এর মাধ্যমেে hello মডিফাই হয়েছে। [প্রয়োজনে আবার খেয়াল করুন।] সব শেষ লাইনে hello() কল হয়েছে। আউটপুট হবে এরকম:



|  |  |
| --- | --- |
| 1  2  3 | Before calling func()  Hello World!  After calling func() |

নাম এবং কাজ দেখে বোঝাই যাচ্ছে mydecorator হচ্ছে আমাদের কাঙ্খিত সেই ডেকরেটর।

তবে ডেকরেটর ব্যাবহারের সুন্দর একটি সিনট্যাক্স আছে, @। উপরের কোড কে আমরা সুন্দর করে এভাবে লিখতে পারি:

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | def mydecorator(func):        def wrapper():          print("Before calling func()")          func()          print("After calling func()")        return wrapper      @mydecorator  def hello():      print("Hello World!")      hello() |

অর্থাৎ, hello = mydecorator(hello) এই লাইনের পরিবর্তে আমরা hello ফাংশনটি ডিফাইনের ঠিক আগে @mydecorator লিখেছি। পূর্বের মত একই কাজ হবে।

### বাস্তব উদাহরণ

এবার একটি বাস্তব উদাহরণ দেখা যাক। মনে করি আমাদের একটি ফাংশন আছে, আমরা চাই যখন এটি কল হবে ঠিক ওই সময় যেন লগ হিসবে একটা ফাইলে থাকে। এর সমাধান দেখে নেয়া যাক:

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | from datetime import datetime  from time import sleep      def log(func):        def wrapper():          with open('log.txt', 'a') as file:              now = datetime.now().strftime("%I:%M:%S%p on %d %B, %Y")              file.write(now + '\n')          func()        return wrapper      @log  def hello():      print('Hello World!')      for i in range(3):      hello()      sleep(5) |

এই প্রোগ্রাম রান করলে কারেন্ট ওয়ার্কিং ডিরেক্টরিতে log.txt নামের একটা ফাইল তৈরি হবে, সেটি খুললে hello() এক্সিকিউট হওয়ার সময় গুলো পাওয়া যাবে।

### প্যারামিটার/আর্গুমেন্ট সহ ডেকরেটর

যদি log ডেকরেটর টায় প্যারামিটার হিসেবে ফাইলের নাম দিয়ে দেয়া যেত, log.txt এর পরিবর্তে আমাদের প্রয়োজন মত নাম, তাহলে সুবিধা হতো না? হ্যা, ডেকরেটরে প্যারামিটার/আর্গুমেন্ট পাস করা সম্ভব। এজন্য আমাদের ডেকরেটরকে আরেকটা ফাংশনের মধ্যে নেস্টেড আকারে রাখা লাগবে। এরকম:

Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | from datetime import datetime  from time import sleep      def log(filename='log.txt'):      def log\_decorator(func):          def wrapper():              with open(filename, 'a') as file:                  now = datetime.now().strftime("%I:%M:%S%p on %d %B, %Y")                  file.write(now + '\n')              func()          return wrapper      return log\_decorator      @log(filename='history.txt')  def hello():      print('Hello World!')      for i in range(3):      hello()      sleep(5) |

এখন আমাদের লগ history.txt খুললে পাওয়া যাবে। আর যদি @log এ কোন প্যারামিটার পাস না করা হয়, তাহলে ডিফল্ট ভাবে log.txt তে লগ থাকবে।

## ডেকোরেটর

নাম শুনেই সবাই বুঝতে পারার কথা কোন কিছুর সৌন্দর্য বর্ধন করার মতই কিছু একটা হবে এখন। ডেকোরেটর হচ্ছে সাধারণ রকমেরই একটা ফাংশন যা অন্য আরেকটি ফাংশনকে মডিফাই করে তথা তার কাজকে বর্ধিত বা পরিবর্তিত করে।

অন্য ভাবে বলতে গেলে, যদি কখনো এমন দরকার পরে যে একটা ফাংশনের ফাংশনালিটি একটু পরিবর্তন/পরিবর্ধন করা দরকার কিন্তু আমরা সেই ফাংশনের কোড পরিবর্তন করতে চাচ্ছি না। তখন ডেকোরেটর ব্যবহার করে আমরা সেই কাজটা করতে পারবো।

একটা উদাহরণ -

>>> def my\_decorator(func):

... def decorate():

... print("--------------")

... func()

... print("--------------")

... return decorate

...

>>> def print\_raw():

... print("Clear Text")

...

>>> decorated\_function = my\_decorator(print\_raw)

>>> decorated\_function()

--------------

Clear Text

ধরুন আমাদের একটা সাধারণ ফাংশন আছে যার নাম print\_raw এবং এটি খুব সাধারণ ভাবেই Clear Text এই বাক্যকে প্রিন্ট করে। এখন আমরা চাই যখনই আমি কোথাও Clear Text বাক্যকে প্রিন্ট করবো সেখানে যেন এর আগে পরে একটু স্টাইল যুক্ত হয় --- চিহ্ন দিয়ে। কিন্তু আবার চাচ্ছি না যে, print\_raw ফাংশনটার কোড পরিবর্তন করতে।

তখন আমি একটি ডেকোরেটর বানালাম যার নাম my\_decorator. এর একটি প্যারামিটার যা হচ্ছে একটি ফাংশন। এই my\_decorator এর মধ্যে আমরা আরেকটি ফাংশন তৈরি করেছি যার নাম decorate.

my\_decorator এর কাছে আসা ফাংশনকে এই decorate ফাংশনটি এক্সিকিউট করে। কিন্তু তার আগে ও পরে দুটি অতিরিক্ত প্রিন্ট স্টেটমেন্ট যোগ করে স্টাইল করে দেয়। পরিশেষে my\_decorator ফাংশনটি এই decorate ফাংশন কে রিটার্ন করে।

এরপর আমরা my\_decorator ফাংশন কে কল করেছি এবং এর আর্গুমেন্ট হিসেবে print\_raw কে পাঠিয়ে দিয়েছি। এটা মডিফাই হয়ে ফিরে এসে decorated\_function ভ্যারিয়েবলে জমা হয়েছে। অতঃপর, decorated\_function() কল করে আমরা Clear Text এর স্টাইলড ভার্সন পাই।

ব্যপারটাকে আরেক্টু সুন্দর করার জন্য আমরা ভ্যারিয়েবল রি-অ্যাসাইন এর সুবিধা নিতে পারি অর্থাৎ -

>>> print\_raw = my\_decorator(print\_raw)

>>> print\_raw()

--------------

Clear Text

এবার মনে হচ্ছে print\_raw এর নাম ধাম ঠিকি আছে শুধু decorate হয়ে এসেছে :)

## ****@decorator****

ধরে নিচ্ছি my\_decorator নামের একটি ডেকোরেটর ডিফাইন করা আছে। এখন আমরা চাইলে আমাদের তৈরি যেকোনো নতুন ফাংশনের উপর একে আপ্লাই করতে পারি। যেমন, আমরা যদি কোথাও নিচের মত একটা ফাংশন লিখি,

def print\_text():

print("Hello World!")

এবং চাই যে এর উপর আমাদের decorator এর স্টাইল আপ্লাই হোক। তাহলে খুব সহজ ভাবে আমরা নিচের মত করে একে একটি decorator এর আওতাধীন করতে পারি,

@my\_decorator

def print\_text():

print("Hello World!")

তাহলে যখনই print\_text কল করা হবে তখনি নিচের মত আউটপুট আসবে,

# ফাংশন(need more study)

যেকোনো রকম কম্পিউটার প্রোগ্রামিং -এ দুটি টার্ম শুনতে পাওয়া যায়। **WET** এবং **DRY**. WET মানে বলা হয় **Write Everything Twice** অথবা **We Enjoy Writing**. অর্থাৎ যেকোনো প্রোগ্রামের মধ্যে অপ্রয়োজনীয় এবং অতিরিক্ত স্টেটমেন্ট লেখা বা ব্যবহার করা যেখানে সেই একই কাজ অনেক কম কোড লিখেও আরও অপ্টিমাইজ ভাবে করা যেত। এটা খারাপ অভ্যাস। আবার DRY মানে বলা হয় - **Do not Repeat Yourself**. অর্থাৎ ওই যে ১০ বার ১০টা লাইন প্রিন্ট করার জন্য ১০ বার প্রিন্ট স্টেটমেন্ট ব্যবহার না করা। যেহেতু কাজটা একই (কিছু একটা প্রিন্ট করা) সেহেতু একবার প্রিন্ট স্টেটমেন্ট লিখেই কোন না কোন ভাবে ১০ বার প্রিন্ট করা সম্ভব। এটা ভালো অভ্যাস।

এরকম কোড পুনব্যবহার করার সবচেয়ে ভালো উদাহরণ হচ্ছে ফাংশন এর প্রয়োগ। আমরা কিন্তু ইতোমধ্যে ফাংশনের ব্যবহার করে ফেলেছি আগের চ্যাপ্টার গুলোতে। যেমন এই যে প্রিন্ট print নিয়ে এতো কথা বলা হল, এটাও একটা ফাংশন। এটা একটা বিল্ট ইন ফাংশন। অর্থাৎ এই ফাংশনের কর্মপ্রণালী আমরা নিজেরা কোড করে লিখি নাই। এই ফাংশনের কাজ হচ্ছে - এর মধ্যে যেকোনো string কে পাঠিয়ে দিলে সেই string -কে সে স্ট্যান্ডার্ড আউটপুটে প্রিন্ট করে দেখায়। অর্থাৎ, কোন কিছু প্রিন্ট করার জন্য আমরা প্রত্যেকবার কম্পিউটারের বোধগম্য এক গাদা কোড লিখি না। শুধু প্রিন্ট ফাংশনকে কল করে যা প্রিন্ট করতে হবে তা ওকে দিয়ে দেই। এভাবে আমরা প্রিন্ট ফাংশনের জন্য করা কোডকে বার বার ব্যবহার করছি। এটাই কোডের পুনব্যবহার।

আমাদের প্রোগ্রামের যে অংশগুলো বার বার আসে সেগুলোকে আমরা পুনরায় ব্যবহারযোগ্য একক (reusable unit) হিসেবে ব্যবহার করতে পারি ফাংশনের সাহায্যে । গনিতে যেমন দেখেছি কোন ফাংশন একটি ইনপুট নিয়ে সেটার উপর বিভিন্ন ধরনের ম্যাথ করে আউটপুট দেয়, প্রোগ্রামিংএও সেই একই ব্যাপার ঘটে । আপনি এক বা একাধিক প্যারামিটার পাস করবেন একটি ফাংশনে, ফাংশনটি প্রসেস করে আপনাকে আউটপুট “রিটার্ন করবে” । তবে প্রোগ্রামিং এর ক্ষেত্রে সবসময় যে ইনপুট থাকতে হবে বা আউটপুট দিতে হবে এমন কোন কথা নেই ।

একটি ফাংশন আসলে কিছু স্টেটমেন্টের সংকলন । যখনই কোন ফাংশন কল করা হয় তখন এই ফাংশনের ভিতরে থাকা স্টেটমেন্টগুলো এক্সিকিউট করা হয় । পাইথনে আমরা ফাংশন ডিক্লেয়ার করার জন্য def কি-ওয়ার্ডটি ব্যবহার করি । আসুন দেখে নেই একটি ফাংশন:

*#example function ,without parameter and print function*

**def** fun():

    print('i am a print type function')

fun()

*#example function , without parameter and return function*

**def** fun1():

**return** 'i am a return type function'

fun1()

print(fun1())

*#example function ,with parameter and print function*

**def** fun2(**x**,**y**):

    print(x**+**y)

fun2(2 ,2)

*#example function , with parameter and return function*

**def** fun3(**x**,**y**):

**return** x**+**y

print(fun3(2,2))

*#example,local variable function*

**def** fun4():

    x**=**2

    y**=**2

    print(x**+**y)

fun4()

**def** fun4(**x**,**y**):

    print(x**+**y)

fun4(2,2)

*#exaple, global variable function*

x**=**2

y**=**2

**def** fun5():

    print(x**+**y)

fun5()

*#example of nested function*

**def** fun1():

    print('i am fun1')

**def** fun2():

        print('i am fun2')

**def** fun3():

            print('i am fun3')

**def** fun4():

                print('i am fun4')

            fun4()

        fun3()

    fun2()

fun1()

### ফাংশন আর্গুমেন্ট

মনে আছে আমরা আগের চ্যাপ্টারে ফাংশনকে একটি ছোট্ট মেশিন হিসেবে কল্পনা করেছিলাম। যেকোনো মেশিন বা যন্ত্র যখন বানানো হয় তখন তার কাজের জন্য যেমন কিছু যন্ত্রপাতির সেটআপ দরকার হয় তেমনি সেই মেশিনে ইনপুট হিসেবে কিছু কাঁচামাল দিতে হয় যেগুলো প্রক্রিয়াজাত করে মেশিন আমাদের চাহিদা মোতাবেক জিনিষ তৈরি করে দেয় বা এর থেকে আউটপুট পাওয়া যায়। ধরে নিচ্ছি আমাদের বানানো মেশিনটির এক পাশ দিয়ে ময়দা, চিনি, দুধ, ক্রিম এসব দিলে আরেক পাশ দিয়ে সুন্দর কেক তৈরি হয়ে বের হয়। তাহলে সেই ময়দা, চিনি, দুধ, ক্রিম এসব হচ্ছে সেই মেশিনের **আর্গুমেন্ট** আর কেক বানানোর জন্য মেশিনের মধ্যে বিভিন্ন যন্ত্রের যে সেটআপ আছে সেটাকে বলা যেতে পারে **ফাংশন বডি**। আর শেষে যে সুস্বাদু কেক পাওয়া যায় তাকে বলা যেতে পারে ফাংশনের **রিটার্ন ভ্যালু**।

def show\_double(x):

print(x\*2)

show\_double(2)

উপরে show\_double ফাংশনের আর্গুমেন্ট একটি.

It means parameter value is argument. আর্গুমেন্টকে ফাংশনের দুটি প্রথম বন্ধনীর মধ্যে ডিফাইন করতে হয়।like, fun(2,2)

একটি ফাংশন কিন্তু একাধিক আর্গুমেন্ট নিয়ে কাজ করতে পারে অর্থাৎ এর একাধিক আর্গুমেন্ট থাকতে পারে। এটাই তো যৌক্তিক, তাই না? কারণ, একটি ফাংশন তথা মেশিনকে জটিল জটিল জিনিষ বানাতে বা আউটপুট দিতে তাকে অনেক গুলো ইনপুট নিয়ে কাজ করতে হতেই পারে। নিচের উদাহরণটি দেখি -

*#locla variable doesnot access outer functuion*

**def** fun(**x**,**y**):

    z**=**x**+**y

    print(z)

fun(2,2)

print(z)

*#global  function acces outer function*

z**=**0

**def** fun(**x**,**y**):

    z**=**x**+**y

    print(z)

fun(2,2)

print(z)

উপরের উদাহরণে, print(z) স্টেটমেন্টটি এরর দেখাচ্ছে কারণ z ভ্যারিয়েবলের গণ্ডি বা স্কোপ ছিল শুধুমাত্র make\_sum ফাংশনের মধ্যেই। তাই বাইরে থেকে একে অ্যাক্সেস করা যায় নি।

### মাল্টিপল প্যারামিটার হ্যান্ডলিং | আর্বিটরারি আর্গুমেন্ট লিস্ট or \*args

মনে করুন, আপনি make\_sum ফাংশনটিতে অনেকগুলো প্যারামিটার পাঠাতে চাচ্ছেন যেমন, 10, 20, 30 ... ইত্যাদি। যদি আপনি make\_sum (a, b) হিসেবে ডিক্লেয়ার করেন তাহলে দুইটার বেশি প্যারামিটার পাঠাতে পারবেন না। সেক্ষেত্রে কোড হবে এইরকম,

*#this function show an error ,cause have two variable but give three argument*

**def** fun(**a**,**b**):

    sum**=**a**+**b

    print(sum)

fun(2,2,2)

*#this function no show error ,cause \*args use*

**def** fun1(**x**,**y**,**\*args**):

    sum**=**x**+**y**+**(args)

    print(sum)

fun1(2,2,2,2,2)

### পাইথনে \* এর অর্থ

\* এর আর্গুমেন্টে ভ্যালু Tuple হিসেবে প্যাকড থাকে। এর মানে \* দিয়ে প্যারামিটার ডিক্লেয়ার করলে আমরা যেকোন সংখ্যক পজিশনাল আর্গুমেন্ট পাস করতে পারি। যেমন করলাম make\_sum এর ক্ষেত্রে। শুরুতে make\_sum মাত্র দুইটা আর্গুমেন্ট নিলেও পরবর্তীতে আমরা প্যারামিটারে \* বসিয়ে দিলাম তখন সে অনেকগুলো আর্গুমেন্ট পাস করতে পারছে।

### পাইথনে \*\* এর অর্থ

আমরা চাইলে ফাংশনের প্যারামিটারে ডাবল অ্যাস্টেরিস্কস বসিয়েও ডিক্লেয়ার করতে পারি। ডাবল স্টারের মানে হল যেকোন সংখ্যক named parameter থাকতে পারে। এই মানগুলো ডিকশনারি হিসেবে প্যাকড থাকে। নিচের উদাহরণটি লক্ষ্য করা যাক,

*#this function show appilication of \*\**

**def** fun2(**\*\*kwargs**):

    print(kwargs)

fun2(**a=**2,**b=**2,**c=**2,)

example by for loop:

**def** print\_dict(**\*\*kwargs**):

**for** args **in** kwargs:

        print("{0} : {1}".format(args, kwargs[args]))

print\_dict(**a=**1, **b=**2, **c=**3)

চাইলে আমরা মিক্সড ভ্যারিয়েডিক আর্গুমেন্ট পাঠাতে পারি। মানে একই ফাংশনে তিন ধরণের আর্গুমেন্ট, তবে খেয়াল রাখতে হবে প্যারামিটারগুলো এমন ভাবে ডিফাইন করা হয় যেন প্রথমে সাধারণ প্যারামিটার তারপরে সিঙ্গেল অ্যাস্টেরিস্কের প্যারামিটার এবং অবশেষে ডাবল অ্যাস্টেরিস্কস এর প্যারামিটার থাকে। মানে আমাদের অবশ্যই ক্রম মানতে হবে এইক্ষেত্রে।

*#application all argument system*

**def** fun5(**x**,**\*args**,**\*\*kwargs**):

    print(x)

    print(args)

    print(kwargs)

fun5(2,5,5,5,5,5,**a=**1,**b=**2,**c=**3)

all type difference

*#this function show an error ,cause have two variable but give three argument*

*#def fun(a,b):*

*#sum=a+b*

*#print(sum)*

*#fun(2,2,2)*

*#this function no show error ,cause \*args use*

**def** fun1(**\*args**):

    print(args)

fun1(2,2,2,2,2)

*#this function show appilication of \*\**

**def** fun2(**\*\*kwargs**):

    print(kwargs)

fun2(**a=**2,**b=**2,**c=**2,)

**def** print\_dict(**\*\*kwargs**):

**for** args **in** kwargs:

        print("{0} : {1}".format(args, kwargs[args]))

print\_dict(**a=**1, **b=**2, **c=**3)

*#application all argument system*

**def** fun5(**x**,**\*args**,**\*\*kwargs**):

    print(x)

    print(args)

    print(kwargs)

fun5(2,5,5,5,5,5,**a=**1,**b=**2,**c=**3)

### প্যারামিটার ও আর্গুমেন্ট

যখন একটি ফাংশনকে ডিফাইন করা হয় তখন এর ভ্যারিয়েবল গুলোকে প্যারামিটার বলা হয়। আর যখন একটি ফাংশনকে কল করা হয় তখন সেই ফাংশনের প্যারামিটার হিসেবে যে ভ্যালু পাঠানো হয় তাকে আর্গুমেন্ট বলা হয়।

### ফাংশন রিটার্ন

খুব গুরুত্বপূর্ণ একটি বিষয় হচ্ছে, যখন কোন ফাংশনের মধ্যে একটি return স্টেটমেন্ট থাকে এবং সেটি এক্সিকিউট হয়, তারপর থেকে ফাংশনের মধ্যে থাকা আর কোন কোড এক্সিকিউট হয় না। অর্থাৎ, ফাংশন তার কাজ শেষে কিছু একটা রিটার্ন করেই থেমে যায়। যেমন –

**def** add\_numbers(**x**, **y**):

    total **=** x **+** y

**return** total

    print("This won't be printed")

print(add\_numbers(4, 5))

উপরের প্রোগ্রামের ফাংশনটির কাজ হচ্ছে দুটো আর্গুমেন্ট ভ্যারিয়েবলকে যোগ করে নতুন একটি ভ্যারিয়েবলে রেখে সেটিকে তার caller এর কাছে রিটার্ন করা । রিটার্ন করেই সে ক্ষান্ত। এরপরে আরেকটি প্রিন্ট স্টেটমেন্ট থাকলেও সেটার কোন এক্সিকিউশন নেই। আর তাই সেই This won't be printed লাইনটিকে স্ক্রিনে দেখা যাচ্ছে না।

### অবজেক্ট হিসেবে ফাংশন

একটা কথা বলা হয় = "পাইথনে সব কিছুই অবজেক্ট"। প্রত্যেকটি অবজেক্টেরই কিছু অ্যাট্রিবিউট ও মেথড থাকে। যেমন, আমরা যখনই একটি স্ট্রিং ভ্যারিয়েবল তৈরি করি a = "Abc" এভাবে, তখন যদি type(a) দেখার চেষ্টা করি তাহলে আউটপুট পাবো <class 'str'> অর্থাৎ এই a অবজেক্টটি str ক্লাসের একটি অবজেক্ট এবং এর কিছু অ্যাট্রিবিউট ও মেথড আছে। যেমন - lower, upper ইত্যাদি, যেগুলো ব্যবহার করে আমরা a কে নিয়ে বিভিন্ন কাজ করতে পারি।

ক্লাস, অবজেক্ট, অ্যাট্রিবিউট, মেথড নিয়ে বিস্তারিত আলোচনা আছে [অবজেক্ট ওরিয়েন্টেড সেকশনে](/oop-python)

ফাংশনও একটি অবজেক্ট অর্থাৎ এরও কিছু অ্যাট্রিবিউট ও মেথড আছে। যেমন একটি ফাংশনকে ডিফাইন করার সাথে সাথেই তার \_\_doc\_\_ নামের অ্যাট্রিবিউট তৈরি হয় যার মাধ্যমে একটি ফাংশনের ডক স্ট্রিং অ্যাক্সেস করা যায়। অন্যান্য সাধারণ ভ্যারিয়েবলের ভ্যালুর মত কোন একটি ফাংশনকেও একটি ভ্যারিয়েবলে অ্যাসাইন বা স্টোর করা যায়।

উদাহরণ,

**def** add\_explanation(**line**):

**return** line **+** '!'

update\_line **=** add\_explanation

print(update\_line("Hello World"))

উপরের প্রোগ্রামে প্রথমে add\_explanation ফাংশনটিকে update\_line ভ্যারিয়েবলে অ্যাসাইন করা হয়েছে। এরপর, যেখানে add\_explanation ফাংশনের দরকার পরেছে সেখানে তাকে update\_line নামে কল করা হয়েছে। এভাবে বস্তুত add\_explanation -টাই কল হচ্ছে। আরেকটু পরীক্ষা করার জন্য আমরা যদি print(update\_line) স্টেটমেন্টটি এক্সিকিউট করি তাহলে আউটপুট আসবে <function add\_explanation at 0x10dbf5668>

তাহলে একটি প্রশ্ন মাথায় আসতে পারে - যেহেতু ফাংশনকে ভ্যারিয়েবলে অ্যাসাইন করা যায় তাহলে কি ভ্যারিয়েবলের মত করে একটা ফাংশনকেও অন্য ফাংশনের আর্গুমেন্ট হিসেবে পাঠানো যাবে? উত্তর হচ্ছে হ্যাঁ। একটা উদাহরণ দেখি,

**def** beautify(**text**):

**return** text **+** '!!!'

**def** make\_line(**func**, **words**):

**return** "Hello " **+** func(words)

print(make\_line(beautify, "world"))

উপরের প্রোগ্রামটি একটু বিশ্লেষণ করা যাকঃ ধরে নিচ্ছি beautify ফাংশনের কাজ হচ্ছে এর কাছে যাই দেয়া হয় তার সাথে তিনটি বিস্ময় চিহ্ন যুক্ত করে রিটার্ন করে। আবার আমাদের একটি ফাংশন আছে make\_line যা দিয়ে একটি বাক্য তৈরি করা হয়। কিন্তু আমরা চাই এর মধ্যে বাক্য তৈরির সময়ই শেষ শব্দের সাথে কিছু বিস্ময় চিহ্ন জুড়ে দিতে। তো, যেহেতু বিস্ময় চিহ্ন জুড়ে দেয়ার ফাংশন আমাদের বানানোই আছে তাই ওই ফাংশনকে make\_line এর একটি আর্গুমেন্ট বা চাহিদা হিসেবে উল্লেখ করতে পারি। অর্থাৎ make\_line কে কল করতে হলে এর আর্গুমেন্ট হিসেবে একটি ফাংশন এবং একটি ডাটা (ধরে নিচ্ছি একটি শব্দ) পাঠাতে হবে। যাতে করে প্রয়োজনে সে ওই beautify ফাংশনকে তার বডির মধ্যে থেকে কল করে ব্যবহার করতে পারে।

make\_line ফাংশনের ডেফিনেশনে এর কাছে আসা ফাংশনকে func নামে রিসিভ করা হয়েছে এবং এর বডির মধ্যে সেই নামেই ব্যবহার করা হয়েছে সাধারণভাবে পাস করা ভ্যারিয়েবলের মত আর তার মাধ্যমে বস্তুত beautify ফাংশন কল হয়েছে।

### কমেন্ট ও ডক স্ট্রিং

পাইথনে যেকোনো কমেন্ট লাইন লেখার আগে তার আগে # (হ্যাস বা পাউন্ড) চিহ্ন ব্যবহার করতে হয়। নিচের উদাহরণটি দেখি –

# few variables below

x = 10

y = 5

# make sum of the above two variables

# and store the result in z

z = x + y

print(z) # print the result

# print (x // y)

# another comment

**ডক স্ট্রিং** কমেন্টের মত আরও একটি জিনিষ হচ্ছে ডক স্ট্রিং বা ডকুমেন্ট স্ট্রিং। কিন্তু এর ব্যবহার একটু ভিন্ন ভাবে আরও স্পেসিফিক ভাবে করা হয়। সাধারণত মাল্টি লাইন স্ট্রিং কে কোন ফাংশনের মধ্যে শুরুতেই লিখে ওই ফাংশন সম্পর্কিত একটি মন্তব্যের ব্লক লেখা হয় যাকে ডক স্ট্রিং বলা হয়।

def greet(word):

"""

Print a word with an

exclamation mark following it.

"""

print(word + "!")

greet("Hello World")

সাধারণ কমেন্ট থেকে এটি একটু আলাদা। যেমন - এই স্ট্রিং গুলোকে প্রোগ্রামের রানটাইমের সময়ও অ্যাক্সেস করা যায় নিচের মত করে

def greet(word):

"""

Print a word with an

exclamation mark following it.

"""

print(word + "!")

# What the fucntion does?

print(greet.\_\_doc\_\_)

# Make sense, now lets use it

greet("Hello World")

উপরের প্রোগ্রামের print(greet.\_\_doc\_\_) স্টেটমেন্টটির মাধ্যমে আমরা greet ফাংশনের ডকুমেন্টেশন দেখে নিয়েছি এবং তারপর একে কল করে ব্যবহার করেছি।

**স্টাইল গাইড**

[গুগল স্টাইল গাইড](https://github.com/google/styleguide) মোতাবেক খুব সুন্দর ভাবে একটি ফাংশনের ডক স্ট্রিং লেখা যেতে পারে নিচের মত করে,

**def** square\_root(**n**):

    """Calculate the square root of a number.

    Args:

        n: the number to get the square root of.

    Returns:

        the square root of n.

    Raises:

        TypeError: if n is not a number.

        ValueError: if n is negative.

    """

**pass**

### ফাংশনাল প্রোগ্রামিং কি

সহজ ভাষায় এটা একটা প্রোগ্রামিং স্টাইল যেটা বিশেষত নির্ভর করে ফাংশনের উপর। higher-order-function গুলো এই ধারার মুল জিনিষ। যে ফাংশন আরেকটি ফাংশনকে আর্গুমেন্ট হিসেবে নিতে পারে এবং অথবা রিটার্ন এলিমেন্ট হিসেবে একটি ফাংশন রিটার্ন করতে পারে তাকে higher-order-function বলা হয়। যেমন –

>>> def make\_twice(func, arg):

... return func(func(arg))

...

>>> def add\_five(x):

... return x + 5

...

>>> print(make\_twice(add\_five, 10))

20

উপরের উদাহরণে, প্রথমেই আমরা make\_twice ফাংশনকে কল করছি আর এর ডেফিনেশন মোতাবেক এর কাছে একটি ফাংশন এবং একটি ভ্যালু পাঠিয়েছি। যে ফাংশন পাঠিয়েছি সেটা হচ্ছে add\_five এবং ভ্যালুটি 10. অন্যদিকে make\_twice ফাংশনের মধ্যে func হিসেবে সেই add\_five ফাংশনকে ক্যাচ করছি এবং একাধিকবার সেটাকে কল করছি। আবার, func তথা add\_five এর একটি আর্গুমেন্ট লাগে। তাই, প্রথমবার এক্সিকিউটের সময় এর মধ্যে 10 কে পাঠিয়ে দিচ্ছি এবং রিটার্ন হয়ে আসছে 15 এবং দ্বিতীয়বার সেই 15 -ই গিয়ে ফাইনালি 20 কে রেজাল্ট হিসেবে পাচ্ছি।

#### পিওর বা শুদ্ধ ফাংশন

ফাংশনাল প্রোগ্রামিং-এর সাথে সাথে পিওর এবং ইম্পিওর ফাংশনের বিষয় চলে আসে। পিওর ফাংশন হচ্ছে সেই ফাংশন যার কোন পার্শ্ব-প্রতিক্রিয়া নাই এবং যে ফাংশন কোন কিছু রিটার্ন করে শুধুমাত্র তার আর্গুমেন্ট ভ্যালুর উপর ভিত্তি করেই। যেমন -

>>> def my\_pure\_function(a,b):

... c = (2 \* a) + (2 \* b)

... return c

...

>>> my\_pure\_function(5,10)

30

উপরের প্রোগ্রামে my\_pure\_function দুটো আর্গুমেন্ট নেয় এবং সেগুলোর উপর কিছু ক্যালকুলেশন করে একটি ভ্যালু রিটার্ন করে। শুরু থেকে শেষ নাগাদ এই ফাংশন বাইরের কোন ভ্যালুর উপর নির্ভর করে না বা বাইরের কোন ভ্যালুকে পরিবর্তনের সাথে সম্পৃক্ত নয়। এটাকে একটা পিওর ফাংশন বলা যেতে পারে।

#### ইম্পিওর বা অশুদ্ধ ফাংশন

>>> my\_list = []

>>> def my\_impure\_function(arg):

... my\_list.append(arg)

...

>>> my\_impure\_function(10)

>>> print(my\_list)

[10]

উপরে my\_impure\_function এর কাছে একটি ভ্যালু গেলে সে তার বাইরে অবস্থান করা একটি লিস্ট যার নাম, my\_list, এর মধ্যে সেই ভ্যালুকে ঢুকিয়ে দিচ্ছে। এই ফাংশনের কাজের গণ্ডি একটু বাড়তি। এটাকে ইম্পিওর ফাংশন বলা হয়।

### ল্যামডা

সাধারণভাবে যখন def কিওয়ার্ড ব্যবহার করে একটি ফাংশন তৈরি করা হয় তখন স্বয়ংক্রিয় ভাবে এটিকে একটি ভ্যারিয়েবলে অ্যাসাইন করে দেয়া হয় যার মাধ্যমে একে পরবর্তীতে কল করা যায়। আবার অন্যদিকে, খুব সহজেই স্ট্রিং বা ইন্টিজার টাইপ ভ্যালুকে কোন রকম ভ্যারিয়েবলে অ্যাসাইন করা ছাড়াও তৈরি করা যায়। ঠিক এই সুবিধাটি (ভ্যারিয়েবলে অ্যাসাইন না করা) ফাংশনের ক্ষেত্রেও উপযোগ করা যায় এবং lambda এর মাধ্যমে। এভাবে তৈরি ফাংশনকে anonymous ফাংশন বলা হয়ে থাকে।

ল্যাম্বডার ব্যবহার খুব ফলপ্রসূ হয় যখন খুব সিম্পল যেমন এক লাইনের একটি ফাংশনকে আরেকটি ফাংশনের আর্গুমেন্ট হিসেবে পাঠানোর দরকার পরে। অর্থাৎ যখন সেই এক লাইনের কাজ করা ফাংশনকে আলাদা ভাবে def দিয়ে ডিফাইন/তৈরি করা অনর্থক মনে হয়।

lambda x,y: x+y - প্রথমে lambda কিওয়ার্ড লিখে এর আর্গুমেন্ট গুলোকে লেখা হয় এবং একটি কোলন দেয়ার পর এই ল্যাম্বডা তথা ফাংশনের কর্মকাণ্ড লেখা হয়। যেমন এই ল্যাম্বডাটি দুটো আর্গুমেন্ট নেয় এবং কাজের কাজ বলতে সেই দুটোকে যোগ করে।

some example:

print((lambda **x**,**y**: x **+** 2 **\*** y)(2,3))

print((lambda **x**,**y**:x**+**y)(2,2))

lol**=**((lambda **x**,**y**:x**+**y)(3,3))

print(lol)

*#not possible*

**def** fun():

    lol**=**((lambda **x**,**y** : x**+**y))

    print(lol)

fun(5,5)

### জেনারেটর

আগের চ্যাপ্টার গুলোতে iterable নিয়ে বেশ কিছু কথা বলা হয়েছে। লিস্ট, টাপল এসব হচ্ছে একধরনের iterable. জেনারেটরও এক রকমের iterable. কিন্তু লিস্ট এর মত এর এলিমেন্ট গুলোকে ইন্ডেক্সিং করা যায় না। কিন্তু তার মানে এই না যে, এর এলিমেন্ট গুলোকে অ্যাক্সেস করা যায় না। বরং for লুপ দিয়ে এর এলিমেন্ট গুলোকেও অ্যাক্সেস করা যায়। সব চেয়ে বড় কথা এলিমেন্ট এর চেইন তৈরি এবং অ্যাক্সেস এক সাথেই করা যায়।

সাধারণ ফাংশন এবং yield স্টেটমেন্ট ব্যবহার করেই এই বিশেষ ধরনের iterable কে তৈরি করা যায়। নিচের উদাহরণ দেখে নেই -

**def** fun():

    i**=**5

**while** i**>**0:

**yield** i

        i**-=**1

**for** i **in** fun():

    print(i)

এখানে my\_iterable() দেখতে একটা সাধারণ ফাংশন। কিন্তু একটু একটু খেয়াল করলে দেখা যাবে, এখানে রিটার্ন এর বদলে yield কিওয়ার্ড ব্যবহার করা হয়েছে। এই ফাংশন খুব সহজ ভাবে while লুপ ব্যবহার করে 5 থেকে 1 পর্যন্ত রিটার্ন (yield) করে। কার কাছে রিটার্ন করে? ওই ফাংশনের নিচেই আমাদের তৈরি একটা for লুপ ওয়ালা স্টেটমেন্টের কাছে। এবং সেই লুপের মধ্যে print ব্যবহার করে এর কাছে আসা রিটার্ন ভ্যালুকে বার বার প্রিন্টও করা যাচ্ছে।

এক্ষেত্রে বলাই যায় যে, লিস্ট এর মত আমাদের my\_iterable() -ও একটা iterable যাকে for লুপ দিয়ে অ্যাক্সেস করে কিছু ভ্যালু পাওয়া যায় যে ভ্যালু গুলো কিনা একটু আগেই আমাদের মত করেই তৈরি।

**গুরুত্বপূর্ণ** একটা ব্যপার খেয়াল করুন, সাধারণ কোন ফাংশনকে বার বার কল করলে সেই ফাংশন বার বার নতুন ভাবে এক্সিকিউট হয় এবং কাজ শেষে নতুন ভ্যালু রিটার্ন করে। কিন্তু এই ক্ষেত্রে একটা মজার ব্যপার ঘটছে। তা হল - যদিও for লুপ দিয়ে বার বার my\_iterable() ফাংশনকে কল করা হচ্ছে কিন্তু ওই ফাংশনের মধ্যে থাকা i এর ভ্যালু কিন্তু ঠিকি সেইভ থাকছে (স্মরণ রাখছে) অর্থাৎ while লুপ টি প্রথমে i এর মান 5 তারপর 4 এভাবে রিটার্ন করছে। এমন না যে, প্রত্যেক বার 5 রিটার্ন হচ্ছে যেভাবে একটা সাধারণ ফাংশনকে একাধিক বার কল করলে হত।

>>> def even\_numbers(x):

... for i in range(x):

... if i % 2 == 0:

... yield i

...

>>> even\_nums\_list = list(even\_numbers(10))

>>> print(even\_nums\_list)

[0, 2, 4, 6, 8]

আবার আসি সেই বিশেষ ধরনের একটা ফাংশন যা একাধারে কিছু ভ্যালু yield করে পক্ষান্তরে একে একটি iterable হিসেবে প্রকাশ করে। এখানে even\_numbers() একটি ফাংশন তথা জেনারেটর (কারণ yield ব্যবহার করছে) যা একটি নির্দিষ্ট রেঞ্জ পর্যন্ত কিছু ভ্যালুর উপর for লুপ দিয়ে অপারেশন চালিয়ে সেখান থেকে শুধু মাত্র জোড় সংখ্যা গুলোকে yield (রিটার্ন) করে। ততক্ষণ পর্যন্ত রিটার্ন করে যতক্ষণ তার কাজের সীমা অর্থাৎ তার কাছে আর্গুমেন্ট হিসেবে আসা ভ্যালুর উপর for লুপ এর অপারেশনের শেষ পর্যন্ত।

### রিকারসন

ফাংশনাল প্রোগ্রামিং -এ রিকারসন খুব গুরুত্বপূর্ণ একটি বিষয়। খুব সহজে বলতে গেলে, রিকারসন হচ্ছে এমন একটা অবস্থা যেখানে একটি ফাংশন নিজেকেই কল করে।

একটা সমস্যা যেটাকে সমাধানের জন্য ছোট ছোট ভাগে ভাগ করা যেতে পারে এবং প্রত্যেকটি ভাগের কাজ আবার অনেকটা একই রকম হবে, সেরকম ক্ষেত্রে রিকারসিভ ফাংশন তথা রিকারসন খুব কাজে লাগে।

**বাস্তব উদাহরণ**

ফ্যাক্টরিয়াল সম্পর্কে অনেকেই জানেন, একটা সংখ্যার ফ্যাক্টরিয়াল মানে হচ্ছে সেই সংখ্যা থেকে শুরু করে তার নিচের ক্রমিক সংখ্যা গুলোর প্রত্যেকটির সামগ্রিক গুণফল। অর্থাৎ, 5 এর ফ্যাক্টরিয়াল = 5x4x3x2x1 = 120

এটাকে এভাবেও চিন্তা করা যায়,

5 এর ফ্যাক্টরিয়াল = 5x(4 এর ফ্যাক্টরিয়াল) = 5x4x(3 এর ফ্যাক্টরিয়াল) = 5x4x3(2 এর ফ্যাক্টরিয়াল) = 5x4x3x2(1 এর ফ্যাক্টরিয়াল) = 5x4x3x2x1

অর্থাৎ প্রত্যেকবার একই কাজ করতে হয় কিন্তু আলাদা আলাদা সংখ্যার জন্য। এবং এই কাজের ফাংশন একটাই হলেই চলে। তাই কি করা যেতে পারে? একই ফাংশনকে বার বার কল করা অর্থাৎ নিজেকে নিজেই একতা নির্দিষ্ট সময় পর্যন্ত কল করা।

**প্রোগ্রাম**

**def** fun(**x**):

**if** x**==**1:

**return** 1

**else**:

**return** x**\***fun(x**-**1)

print(fun(5))

উপরের প্রোগ্রামটি দিয়েই যেকোনো সংখ্যার ফ্যাক্টরিয়াল বের করা সম্ভব। এখানে ফাংশনের শুরুতেই চেক করা হয়েছে যে সংখ্যার ফ্যাক্টরিয়াল বের করতে হবে সেটি 1 কিনা। যদি তাই হয় তাহলে ফ্যাক্টরিয়াল 1 এর মান 1 রিটার্ন করা হচ্ছে। এই অবস্থায় রিকারসন থেমে যায়। এটাকে **বেইজ কেস** বলা হয়।

এই কন্ডিশন মিথ্যা হলে আরেকটি জিনিষ রিটার্ন করা হয়। কি রিটার্ন করা হয় সেটাই মজার। রিটার্ন করা হচ্ছে সেই সংখ্যা এবং তার সাথে গুন আকারে ঠিক এই ফাংশনকেই (কল) শুধু আর্গুমেন্ট হিসেবে এক ক্রম কমিয়ে দিয়ে। এভাবে ঘটনা ক্রমে এবং প্রয়োজন অনুসারে একটি ফাংশন নিজেই নিজেকে কল করছে যেটাকেই রিকারসন বলা হয়।

**বেইজ কেস এর গুরুত্ব**

নিচের প্রোগ্রামে কোন বেইজ কেস নাই কিন্তু একটি ফাংশন নিজেই নিজেকে কল করছে। অর্থাৎ এর কল থামার কোন লজিক সেট করা হয় নাই। এটা অনন্তকাল পর্যন্ত চলার চেষ্টা করবে।

**def** factorial(**x**):

**return** x **\*** factorial(x**-**1)

print(factorial(5))

**ডিরেকশন বা দিক**

রিকারসন যেকোনো দিকেই ঘটতে পারে। অর্থাৎ প্রথম একটি ফাংশন আরেকটি দ্বিতীয় ফাংশনকে কল করতে পারে আবার সেই দ্বিতীয় ফাংশন প্রথেম ফাংশনকে কল করতে পারে যেটা কিনা আবার দ্বিতীয় ফাংশনকে কল করতে পারে।

def is\_even(x):

if x == 0:

return True

else:

return is\_odd(x-1)

def is\_odd(x):

return not is\_even(x)

print(is\_odd(17))

print(is\_even(23))

### itertools

এটি পাইথনের একটি স্ট্যান্ডার্ড মডিউল যার বেশ কিছু ফাংশন ব্যবহৃত হয় ফাংশনাল প্রোগ্রামিং এর সময়। যেমন, count ফাংশন একটি নির্দিষ্ট ভ্যালু থেকে ইনফিনিট পর্যন্ত হিসাব করে। cycle ফাংশন একটি iterable কে ইনফিনিট পর্যন্ত ইটারেট করে। repeat ফাংশন ইনিফিনিট অথবা একটি নির্দিষ্ট পরিমাণ পর্যন্ত একটি অবজেক্টকে রিপিট করে।

from itertools import count

for i in count(3):

print(i)

if i >= 11:

break

[ম্যাপ ও ফিল্টার](http://python.howtocode.com.bd/map-filter.html) যেমন কোন ইটারেবল এর উপর কাজ করে তেমনি itertools এর বেশ কিছু ফাংশন যেকোনো রকম iterable যেমন লিস্ট, ডিকশনারি এর উপর কাজ করতে সাহায্য করে। যেমন accumulate ফাংশনের মাধ্যমে একটি লিস্টের সব গুলো ভ্যালুর রানিং টোটাল পাওয়া সম্ভব।

from itertools import accumulate

my\_numbers = [1, 2, 3, 4, 5, 6]

accumulated\_numbers = accumulate(my\_numbers)

list\_of\_accu\_nums = list(accumulated\_numbers)

print(list\_of\_accu\_nums)

আরেকটি মজার ফাংশন takewhile যার নাম শুনেই বোঝা যাচ্ছে এটা কিছু সময় পর্যন্ত কিছু একটা নিয়ে নেয়। আর আগেই বলা হয়েছে এর অপারেশন হতে পারে যেকোনো ইটারেবলের উপর। এটা সেই সব ভ্যালুকে বের করে নেয় যেগুলোর জন্য একটি নির্দিষ্ট প্রেডিকেট সত্য হয়। নিচের উদাহরণে lambda x: x <= 6 ল্যাম্বডাটি একটি প্রেডিকেট। ল্যাম্বডা নিয়ে পড়তে হবে [এখানে](http://python.howtocode.com.bd/lambda.html)

from itertools import takewhile

my\_numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

nums\_less\_equal\_six = takewhile(lambda x: x <= 6, my\_numbers)

filtered\_numbers = list(nums\_less\_equal\_six)

print(filtered\_numbers)

আরও ফাংশন এবং উদাহরণ,

from itertools import product, permutations

letters = ("A", "B")

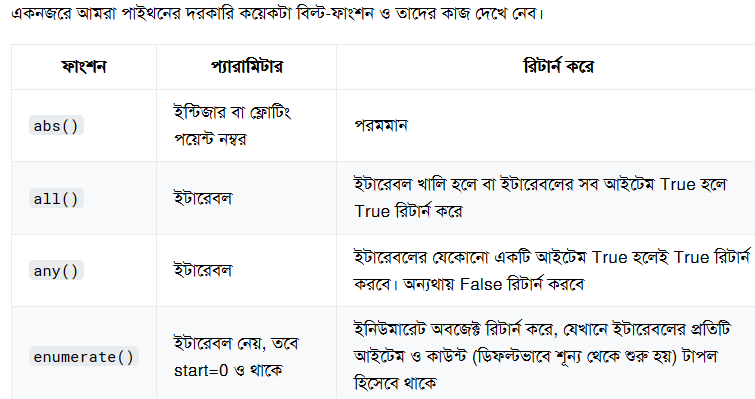
print(list(product(letters, range(2))))

print(list(permutations(letters)))

### type of function

সাধারণত ফাংশনকে আমরা দুই ভাগে ভাগ করতে পারি : ১. বিল্ট-ইন (built-in) ফাংশন ও ২. ইউজার-ডিফাইন্ড (user-defined) ফাংশন।

এতক্ষণ আমরা যেসব ফাংশন ব্যবহার করেছি যেমন : print(), input(), pop(), এগুলো সবই হলো বিল্ট-ইন ফাংশন। পাইথনের সাথে রেডিমেড পাওয়া যায় বলে আমরা এগুলোকে বিল্ট-ইন ফাংশন বলি।



### ফাংশন তৈরি ও কল করা

পাইথনে ফাংশন তৈরির বা লেখার নির্দিষ্ট নিয়ম কানুন আছে।

* ফাংশন ব্লক def কি-ওয়ার্ড দিয়ে শুরু হবে। def এর পর একটা স্পেস দিয়ে ফাংশনের নাম থাকবে। ফাংশনের নাম ভ্যারিয়েবলের নামের মতই হতে পারে। তবে ফাংশনের নাম আন্ডারস্কোর \_ দিয়ে শুরু করা যায়।
* ফাংশনের নামের পর ১ম ব্রাকেট () থাকবে। ব্রাকেটের ভিতর এক বা একাধিক প্যারামিটার (parameter) বা আর্গুমেন্ট (argument) কমা দিয়ে সেপারেট করা থাকবে। ব্রাকেটের পর কোলন : চিহ্ন থাকবে।
* ফাংশনের সব স্টেটমেন্ট ইনডেন্টেড থাকবে। প্রথম স্টেটমেন্টে স্রেফ একটা কমেন্ট থাকলে ভাল হয়। এই কমেন্টে ফাংশনের একটা সংক্ষিপ্ত পরিচয় থাকবে। তবে এটা বাধ্যতামূলক নয়।
* ফাংশন শেষ হবে return কি-ওয়ার্ড দিয়ে। কোন ডাটা রিটার্ন করার ক্ষেত্রে তার কি-ওয়ার্ডের পর একটা স্পেস দিয়ে থাকবে।

def dhan\_katar\_machine(dhan):

# Eta ekta dhan katar machine

.............................

.............................

return chal

একটা জিনিস খেয়াল করেছি আমরা? return কি-ওয়ার্ডের পরে আমরা কিন্তু কোন ভ্যারিয়েবল বা ভ্যালু দিই নাই। এক্ষেত্রে ফাংশন None রিটার্ন করবে। আসলে শুধু return আর return None একই কথা।

**def** fun():

    print('i am print function')

**return**

fun()

### ফাংশন প্যারামিটার বা আর্গুমেন্ট

ফাংশনে এই যে প্যারামিটার বা আর্গুমেন্ট আমরা ব্যবহার করতেছি এদেরকে মোটামুটি চারভাগে ভাগ করতে পারি।

* রিক্যুয়ার্ড আর্গুমেন্ট (Required argument)
* কি-ওয়ার্ড আর্গুমেন্ট (Keyword argument)
* ডিফল্ট আর্গুমেন্ট (Default argument)
* ভ্যারিয়েবল লেংথ আর্গুমেন্ট (Variable-length argument)

চলুন এদের সম্পর্কে আরেকটু বিস্তারিত জানা যাক।

#### রিক্যুয়ার্ড আর্গুমেন্ট (Required argument)

একটা উদাহরণ দেখা যাক আগে।

**def** fun(**x**,**y**,**z**):

**return** x**+**y**+**z

print(fun(2,2))

আউটপুটটা খেয়াল করি সবাই। পাইথন TypeError থ্রো করেছে - একটা রিক্যুয়ার্ড পজিশনাল আর্গুমেন্ট মিসিং। আমাদের ফাংশনটার আসলে তিনটা প্যারামিটার নিয়ে কাজ করার কথা। কিন্তু আমরা দিয়েছি দুইটা, তাই তৃতীয়টা ছাড়া ফাংশনটা কাজ করছে না। এক্ষেত্রে তৃতীয়টা হল ফাংশনটার রিক্যুয়ার্ড আর্গুমেন্ট।

#### কি-ওয়ার্ড আর্গুমেন্ট (Keyword argument)

একটা উদাহরণ দেখা যাক

**def** fun(**a**,**b**,**c**):

**return** a**+**b**+**c

print(fun(**a=**2,**c=**5,**b=**3))

এবার আমরা ফাংশনে ভ্যালু পাস করেছি এভাবে: b=2, c=3, a=1। পজিশনাললি নয়, বরং কোনটার মান কত হবে তা সরাসরি বলে দিয়েছি। এভাবে কি-ওয়ার্ড আর্গুমেন্ট ব্যবহার করার একটা সুবিধা রয়েছে, বিশেষ করে অনেকগুলো আর্গুমেন্ট ব্যবহার করার সময়, আর্গুমেন্টের ধারাবাহিকতা রক্ষা করার দরকার পড়ে না।

#### ডিফল্ট আর্গুমেন্ট (Default argument)

একটা উদাহরণ দেখা যাক।

**def** fun(**x**,**y**,**z=**3): *#defoult argument*

**return** x**+**y**+**z

print(fun(3,3))

তিন নাম্বার লাইনটা বিবেচনা করা যাক। আমরা কিন্তু প্যারামিটার হিসাবে এবার দুইটা ভ্যালু পাস করেছি। অথচ ফাংশনটা তিনটা প্যারামিটার নেয়। তারপরও কোন এরর থ্রো করে নাই। কারণ হল, আমরা c এর ভ্যালু ডিফল্টভাবে 3 দিয়ে দিয়েছি। এতে যদি c এর জন্য কোন ভ্যালু পাস করা না হয় তবে ডিফল্ট ভ্যালু 3 নিয়েই ফাংশন কাজ করবে। আর যদি কোন ভ্যালু পাস করা হয় তবে ঐ ভ্যালুটা নিয়ে ফাংশন কাজ করবে। আরেকটা উদাহরণ দিলেই ব্যাপারটা পরিষ্কার হয়ে যাবে।

**def** fun(**x**,**y=**3,**z**): *#this type is not possible ,cause defoult argument come from last to first*

**return** x**+**y**+**z

print(fun(3,3))

#### \*args ভ্যারিয়েবল লেংথ আর্গুমেন্ট

**def** fun(**\*args**):

    x**=**0

**for** lol **in** args:

        x**=**x**+**lol

**return** x

print(fun(2,2,2,2,2,2,2,))

ফাংশনের যেকোন প্যারামিটারের আগে একটা অ্যাসটেরিস্ক \* চিহ্ন দিলে সেটা আনলিমিটেড ভ্যালু হোল্ড করতে পারে। জেনে রাখা ভাল, এই প্যারামিটারটা একটা টাপল তৈরি করে সবগুলো ভ্যালু হোল্ড করে। পরে একটা for লুপ চালিয়ে আমরা সবগুলো ভ্যালু অ্যাক্সেস করতে পারি। এটাকে নন-কীওয়ার্ডেড (non-keyworded) ভ্যারিয়েবল লেংথ আর্গুমেন্ট বলা হয়।

যদি আমরা কি-ওয়ার্ড আর্গুমেন্ট পাস করতে চাইতাম? তখন প্যারামিটারের আগে দুইটা অ্যাসটেরিস্ক \* চিহ্ন দিতে হত। এটাকে কীওয়ার্ডেড (keyworded) ভ্যারিয়েবল লেংথ আর্গুমেন্ট বলা হয়। এই প্যারামিটারটা একটা ডিকশনারি তৈরি করে সবগুলো ভ্যালু হোল্ড করে। যেমন:

#### \*\*kwargs

**def** fun(**\*\*kwargs**):

**return** kwargs

print(fun(**a=**2,**y=**3,**z=**4))

#### local and global variable

একটা জিনিস খেয়াল করেছি - শেষের দুই উদাহরণে আমরা ফাংশনের ভিতরেও কিন্তু ভ্যারিয়েবল ডিক্লেয়ার করেছি। এটাকে বলা হয় লোকাল ভ্যারিয়েবল (Local variable)। ফাংশনের ভিতরে ডিক্লেয়ার করা সব ফাংশনই লোকাল ভ্যারিয়েবল আর এদেরকে শুধু ফাংশনের ভিতরেই অ্যাক্সেস করা যায়। অন্যদিকে, ফাংশনের বাহিরে ডিক্লেয়ার করা ভ্যারিয়েবলগুলোকে বলা হয় গ্লোবাল ভ্যারিয়েবল (Global variable)। গ্লোবাল ভ্যারিয়েবল প্রোগ্রামের যেকোন জায়গা থেকেই অ্যাক্সেস করা যায়। লোকাল ও গ্লোবাল ভ্যারিয়েবল নিয়ে সামনের চাপ্টারগুলোতে আমরা আরো কিছু শিখব।

### ???রিকার্সন (Recursion)

রিকার্সন একটা মজার জিনিস। যখন কোন ফাংশন নিজেই নিজেকে ডাকে তখন আমরা তাকে রিকার্সন বলি। আর ঐ ফাংশনটাকে বলি রিকার্সিভ (Recursive) ফাংশন। নিজেকে নিজে ডাকে মানে হল নিজের ভিতরেই আবার নিজেকে কল করবে। একটা সিম্পল উদাহরণ দেখা যাক।

**def** counter(**num**):

    print(num)

    num **+=** 1

    counter(num)

counter(1)

এখানে counter() ফাংশন প্যারামিটার হিসাবে 1 নিয়েছে। তারপর 1 কে প্রিন্ট করে এর মান 1 বাড়িয়ে দেয়। তারপর নিজেই নিজেকে কল করে আর এইবার প্যরাামিটার হিসাবে বাড়ানো সংখ্যাটা মানে 2 পাস করা হয়। ফলাফল কি হবে? ফাংশনটা আবার এক্সিকিউট হওয়া শুরু করবে। আর এবার সে 2 প্রিন্ট করবে। তারপর 2 এর মান 1 বাড়িয়ে 3 করবে। শেষমেষ আবার নিজেকে কল করবে ওর 3 পাস করবে প্যারামিটার হিসাবে। এভাবে কিন্তু চলতেই থাকবে। আমরা এই জিনিসটাকেই বলছি রিকার্সন।

একটু আগে রিকার্সনের একটা বাকওয়াশ উদাহরণ দেখেছি আমরা। এবার আমরা একটা কাজের উদাহরণ দেখব। তবে তার আগে জানব ফ্যাক্টোরিয়াল (Factorial) সম্পর্কে।

ফ্যাক্টোরিয়াল একটা গাণিতিক টার্ম বা ফাংশন। সহজ ভাষায়, ফ্যাক্টোরিয়াল হল এমন একটি ফাংশন যা একটি সংখ্যাকে এর নিচের সবগুলো সংখ্যা (১ পর্যন্ত) দিয়ে গুণ করে। আর কোন সংখ্যার ফ্যাক্টোরিয়াল বোঝানোর জন্য ! চিহ্ন ব্যবহার করা হয়। যেমন: 5! মানে হল 5 এর ফ্যাক্টোরিয়াল বা 5\_4\_3\_2\_1=120। তবে 0! এর মান সবসময় 1 হবে, মুখস্থবিদ্যা। মোটামুটি বোঝা গেছে নাকি! তাহলে এবার আমরা ফ্যাক্টোরিয়াল বের করার জন্য একটা প্রোগ্রাম লিখব।

Example

print('Please input your number:')

number = int(input())

temp = number

while number > 1:

number -= 1

temp = temp\*number

if temp == 0:

print(1)

else:

print(temp)

একটা লুপ ব্যবহার করে আমরা রিকার্সন বের করার প্রোগ্রাম লিখলাম। সিম্পল লজিক ব্যবহার করেছি। লুপ ঘুরিয়ে ঘুরিয়ে আর মান কমিয়ে কমিয়ে গুণ করেছি। একেবারে বিশুদ্ধ বাংলা গুণ। আর এই একই কাজ এবার করব রিকার্সিভ ফাংশন ব্যবহার করে।

def factorial(number):

if number == 0:

return 1

else:

return number \* factorial(number - 1)

print('Please input your number:')

number = int(input())

print(factorial(number))

**আউটপুট**

0

1

5

120

কাজ কিন্তু একই হল। এখানে factorial() হল রিকার্সিভ ফাংশন। কারণ এর রিটার্ন অংশে গিয়ে আমরা আবার একেই কল করেছি। তবে আগের চেয়ে কাজটা কিন্তু সহজ আর সুন্দর হয়েছে অনেক।

return number \* factorial(number – 1) এই বিখ্যাত লাইনটা আমাদের কারও কারও কাছে একটু বিটখিটে লাগতে পারে। মনে প্রশ্ন জাগতে পারে যে আসলে কী ঘটছে সেখানে। ব্যাপারটা একটু বিশ্লেষণ করা যাক। আমরা ধরে নিলাম, factorial() ফাংশনটির মধ্যে ভ্যালু হিসেবে আমরা 5 পাস করছি। 5 যেহেতু 0-এর সমান নয়, তাই if ব্লক এক্সিকিউট না হয়ে else ব্লক এক্সিকিউট হবে। এবার আসা যাক, ওই বিখ্যাত লাইনটায়। এই লাইনটায় আমরা আবার factorial() ফাংশনটাকে কল করেছি। তবে এবার ভ্যালু ১ কমিয়ে 4 পাস করেছি। আর এর সাথে number-এর আগের ভ্যালু 5 গুণ আকারে তো আছেই। সুতরাং factorial() ফাংশনটা প্যারামিটার 4 নিয়ে আবার এক্সিকিউট হওয়া শুরু করবে। 4 যেহেতু 0-এর সমান নয়, তাই if ব্লক এক্সিকিউট না হয়ে else ব্লক এক্সিকিউট হবে।

এই ব্লকে এসে আমরা আবার factorial() ফাংশনটাকে কল করেছি। যথারীতি ভ্যালু ১ কমিয়ে 3 পাস করেছি। আর এর সাথে number-এর আগের ভ্যালু 4 গুণ আকারে আছে। পাশাপাশি শুরুর 5ও গুণ আকারে আছে। অতঃপর factorial() ফাংশনটি প্যারামিটার 3 নিয়ে আবার এক্সিকিউট হওয়া শুরু করবে। এভাবে চলতে চলতে যখন ফাংশনটির মধ্যে 1 পাস হবে, তখন সেটা থেমে যাবে। কিন্তু প্রশ্ন হলো দিন শেষে ফাংশনটি কি ভ্যালু রিটার্ন করবে?

5 ইনপুট দেওয়ার পরে ফাংশন এক্সিকিউট হওয়া শুরু হলো—

* ১ম বার : 5 \* factorial(4)
* ২য় বার : 5 \* 4 \* factorial(3)
* ৩য় বার : 5 \* 4 \* 3 \* factorial(2)
* ৪র্থ বার : 5 \* 4 \* 3 \* 2 \* factorial(1)
* ৫ম বার : 5 \* 4 \* 3 \* 2 \* 1 \* factorial(0)

factorial(0) হলে কিন্তু if ব্লক এক্সিকিউট হয়ে 1 রিটার্ন করে।

দিন শেষে : 5 \* 4 \* 3 \* 2 \* 1 \* 1

গুণ করলে 120 পাওয়া যায়। আর দিনশেষে এটাই রিটার্ন হবে।

### এক লাইনের ফাংশন - ল্যাম্বডা (lambda)

lambda অপারেটর ইউজ করে পাইথনে এক লাইনের ফাংশন লেখা যায়। lambda এর পর স্পেস দিয়ে আর্গুমেন্ট দিতে হয়। তারপর কোলন : চিহ্ন দিয়ে অ্যারিথমেটিক এক্সপ্রেশন দিতে হয়। ফাংশনটাকে একটা নাম দেয়ার জন্য যেকোন ভ্যারিয়েবলে অ্যাসাইন করা যেতে পারে।

sum = lambda a, b : a + b

print(sum(10, 20))

print((lambda a, b : a + b)(10, 20))

**আউটপুট**

30

30

শেষ লাইনে আমরা ফাংশনটাকে কোন ভ্যারিয়েবলে অ্যাসাইন না করেই আমাদের কাজ সিদ্ধ করেছি। আসলে ল্যাম্বডার সুবিধাটাই এখানে। যখন এক লাইনের একটা ফাংশনকে def দিয়ে পয়দা করা অনর্থক মনে হয় (যেমন: যখন একটা ফাংশনকে আরেকটা ফাংশনের আর্গুমেন্ট হিসাবে পাঠানোর দরকার হয়) তখন ল্যাম্বডা ব্যবহার করতে হয়। ওহ! যে কথা হয়নি বলা - এইরকম ফাংশনকে অ্যানোনিমাস (anonymous) ফাংশন বলা হয়। আরেকটা উদাহরণ দেখা যাক:

def my\_function(func, arg1, arg2):

return func(arg1, arg2)

print(my\_function(lambda a, b : a + b, 10, 20))

**আউটপুট**

30

এখানে my\_function() আর্গুমেন্ট হিসাবে একটা ফাংশন আর দুইটা ভ্যালু নিয়েছে। ফাংশন হিসাবে আমরা একটা ল্যাম্বডা ফাংশন এবং ভ্যালু হিসাবে 10, 20 কে পাস করেছি। আর তারপর my\_function() ল্যাম্বডা ফাংশনকেই রিটার্ন করেছে। ভ্যালু দুইটাকে ল্যাম্বডা ফাংশনের আর্গুমেন্ট হিসাবে কাজে লাগানো হয়েছে।

### ??map()

map() একটা বিল্ট-ইন ফাংশন। কিন্তু ফাংশন (বিল্ট-ইন বা ইউজার ডিফাইন্ড) অ্যাপ্লাই করার ক্ষেত্রে এর ব্যবহার ব্যাপক। ম্যাপ কাজ করার জন্য দুইটা আর্গুমেন্ট নেয় - প্রথমটা হল ফাংশন আর দ্বিতীয়টা হল ইটারেটর অবজেক্ট। এর কাজ হল ইটারেটর অবজেক্টের প্রতিটা আইটেমের উপর আর্গুমেন্ট হিসাবে নেয়া ফাংশনটাকে অ্যাপ্লাই করবে। কি বুঝতে পারলাম না? একটা উদাহরণ দেখা যাক।

my\_list = [2, 3, 4, 5, 6, 7]

def square(x):

return x \* x

new\_list = map(square, my\_list)

print(new\_list)

print(list(new\_list))

**আউটপুট**

<map object at 0x7f37c8e7df28>

[4, 9, 16, 25, 36, 49]

এখানে my\_list লিস্টের প্রতিটা আইটেমের উপর square() ফাংশনটা অ্যাপ্লাই করা হয়েছে। new\_list লিস্টের ভিতর একটা ম্যাপ অবজেক্ট স্টোর হয়েছে। পরে আমরা এটাকে লিস্ট হিসাবে কাস্ট করে প্রিন্ট করে দেখেছি।

এ তো একটা ইউজার ডিফাইন্ড ফাংশনকে অ্যাপ্লাই করলাম। এবার আমরা একটু বিল্ট-ইন ফাংশন নিয়ে খেলব।

>>> a, b = map(int, input().split())

10 20

>>> type(a)

<class 'int'>

>>> type(b)

<class 'int'>

>>> a + b

30

আগের ব্যাখ্যাটা বুঝে থাকলে এই প্রোগ্রামটা বোঝা উচিত সবার।

### ??filter()

অনেকটা ম্যাপ ফাংশনের মতই। তবে এর কাজ হল ফিল্টারিং করা। সত্য-মিথ্যার উপর ভিত্তি করে ফিল্টারিং করে। ফিল্টার কাজ করার জন্য দুইটা আর্গুমেন্ট নেয় - প্রথমটা হল ফাংশন আর দ্বিতীয়টা হল ইটারেটর অবজেক্ট। এর কাজ হল ইটারেটর অবজেক্টের প্রতিটা আইটেমের উপর আর্গুমেন্ট হিসাবে নেয়া ফাংশনটাকে অ্যাপ্লাই করবে। আর যে আইটেমের কারণে ফাংশন False রিটার্ন করবে সেটাই ইটারেটর অবজেক্ট থেকে বাদ। কি বুঝতে পারলাম না? একটা উদাহরণ দেখা যাক।

my\_list = [2, 3, 4, 5, 6, 7]

def is\_even(x):

if (x % 2) == 0:

return True

else:

return False

new\_list = filter(is\_even, my\_list)

print(new\_list)

print(list(new\_list))

**আউটপুট**

<filter object at 0x7f5d35eecef0>

[2, 4, 6]

এখানে my\_list লিস্টের প্রতিটা আইটেমের উপর is\_even() ফাংশনটা অ্যাপ্লাই করা হয়েছে। new\_list লিস্টের ভিতর একটা ম্যাপ অবজেক্ট স্টোর হয়েছে। পরে আমরা এটাকে লিস্ট হিসাবে কাস্ট করে প্রিন্ট করে দেখেছি। কি দেখলাম? দেখলাম যেগুলো জোড় সংখ্যা শুধু সেগুলোই আছে এখন লিস্টে। বাকিগুলো ফিল্টার হয়ে বিদায় নিয়েছে।

এই ছিল ফাংশন নিয়ে আমাদের আলোচনা। তবে শুধু আলোচনাতে সীমাবদ্ধ থাকলেই চলবে না। চর্চা করে ফাংশনকে মনের ভিতর গেঁথে নিতে হবে।

### প্রব্লেম-১

ইউজার এক লাইনে তিনটা পূর্ণসংখ্যা ইনপুট দেবে (যেমন: 23 10 96)। সবচেয়ে বড় সংখ্যাটি বের করতে হবে।

print("Please input three integers:")

a, b, c **=** map(int, input().split())

**if** a **>=** b **and** a **>=** c:

    greatest **=** a

**elif** b **>=** a **and** b **>=** c:

    greatest **=** b

**else**:

    greatest **=** c

print(greatest)

### প্রব্লেম-২:

ছোটবেলায় স্কুলে আমরা গ.সা.গু বা গরিষ্ঠ সাধারণ গুণনীয়কের অনেক সমস্যা সমাধান করেছি। ইংরেজিতে একে বলা হয়, GCD বা Greatest Common Divisor। দুটি সংখ্যার GCD বের করার জন্য একটি ফাংশন লিখতে হবে।

def gcd(a, b):

if b > a:

gcd(b, a)

while b != 0:

temp = a%b

a = b

b = temp

return a

print("Please input two integers:")

a, b = map(int, input().split())

print(gcd(a, b))

দুটি সংখ্যার গ.সা.গু. বের করার জন্য ছোটবেলায় আমরা ইউক্লিডীয় পদ্ধতি ব্যবহার করতাম। বেশ সহজ আর স‌োজাসাপটা পদ্ধতি। দুটি সংখ্যার মধ্যে অপেক্ষাকৃত ছোট সংখ্যাটিকে ভাজক আর বড় সংখ্যাটিকে ভাজ্য ধরে নিয়ে ভাগ করতে হবে। যদি ভাগশেষ শূন্য হয় তাহলে খেলা শেষ। না হলে, ভাগশেষকে ভাজক আর একটু আগের ভাজককে ভাজ্য ধরতে হবে। এবার আবার ভাগ করতে হবে। এবার যতক্ষণ না পর্যন্ত ভাগশেষ শূন্য হয় ততক্ষণ, খেলা চলতে হবে। ভাগশেষ শূন্য হলেই খেলা শেষ এবং সেই ধাপের ভাজকই হবে গ.সা.গু.।

এটুকু ঠিকমত বুঝে থাকলে gcd() ফাংশনটা বোঝা কোন ব্যাপারই না। while লুপের ব্লকে a কে b দিয়ে ভাগ করে ভাগশেষ temp ভ্যারিয়েবলে অ্যাসাইন করেছি। তারপর a ভ্যারিয়েবলে b এবং b ভ্যারিয়েবলে temp ভ্যারিয়েবলের ভ্যালু অ্যাসাইন করেছি। যতক্ষণ না পর্যন্ত b এর ভ্যালু শূন্য হয়, ততক্ষন এই খেলা চালিয়ে গিয়েছি আমরা। b এর ভ্যালু শূন্য হলেই লুপ থেকে বের হয়ে গিয়ে a ভ্যারিয়েবলকে রিটার্ন করেছি। এই a ভ্যারিয়েবলের ভ্যালুই হল আমাদের বহু আকাঙ্ক্ষিত গ.সা.গু।

একটা খুব-খুব-খুবই সহজ কুইজ – b যদি a ভ্যারিয়েবলের চাইতে বড় হয় তাহলে ঘটনা কি ঘটবে?

### প্রব্লেম-৩:

গ.সা.গু তো হয়ে গেল। ল.সা.গু বা লঘিষ্ঠ সাধারণ গুণিতকই বা বাদ থাকবে কেন? ইংরেজিতে একে বলা হয় LCM বা Least Common Multiple। দুটি সংখ্যার LCM বের করার জন্য একটি ফাংশন লিখতে হবে।

def gcd(a, b):

if b > a:

gcd(b, a)

while b != 0:

temp = a%b

a = b

b = temp

return a

def lcm(a, b):

return (a\*b)//gcd(a, b)

print("Please input two integers:")

a, b = map(int, input().split())

print(lcm(a, b))

**আউটপুট**

Please input two integers:

12 18

36

Please input two integers:

96 13

1248

দুটি সংখ্যার ল.সা.গু. বের করার একটা চমৎকার সূত্র আছে। দুটি সংখ্যার গুণফলকে ঐ দুটি সংখ্যার গ.সা.গু. দিয়ে ভাগ করলে সংখ্যা দুটির ল.সা.গু. পাওয়া যায়। একদম ‘জলবৎ তরলং’ টাইপের সমস্যা।

### প্রব্লেম-৪:

ইউজার একটি পূর্ণসংখ্যা ইনপুট দেবে। আমাদের বলতে হবে সেটি মৌলিক (Prime) সংখ্যা, নাকি যৌগিক (Composite) সংখ্যা।

০ আর ১ ব্যতীত যেসব সংখ্যা ওই সংখ্যা আর ১ ব্যতীত অন্য কোনো সংখ্যা দিয়ে নিঃশেষে বিভাজ্য হয় না, তাদের মৌলিক সংখ্যা বলে।

def is\_prime(n):

if n <= 1:

raise ValueError('The number must be greater than 1.')

elif n <= 3:

return True

elif (n % 2) == 0 or (n % 3) == 0:

return False

else:

i = 5

while (i \* i) <= n:

if (n % i) == 0 or (n % i+2) == 0:

return False

i = i + 6

return True

print('Please input your number:')

number = int(input())

if is\_prime(number):

print(number, 'is a prime number.')

else:

print(number, 'is a composite number.')

**আউটপুট**

Please input your number:

5

5 is a prime number.

Please input your number:

18

18 is a composite number.

আমরা এখানে is\_prime() নামে একটা ফাংশন লিখেছি। ফাংশনটা একটা সংখ্যাকে চেক করে দেখে যে তা মৌলিক সংখ্যা কিনা।

একটা সংখ্যা মৌলিক কিনা তা টেস্ট করাকে ইংরেজিতে প্রাইমালিটি টেস্ট (Primality test) বলা হয়। বাংলায় আমরা একে ‘মৌলিকতা পরীক্ষা’ বলতে পারি। মৌলিকতা পরীক্ষার অনেকগুলো পদ্ধতি রয়েছে: ট্রায়াল ডিভিশন মেথড, ফার্মার প্রাইমালিটি টেস্ট, মিলার-রবিন প্রাইমালিটি টেস্ট, সলোভা-স্ট্রাসেন প্রাইমালিটি টেস্ট, একেএস প্রাইমালিটি টেস্ট ইত্যাদি। আমাদের is\_prime() ফাংশনটি ট্রায়াল ডিভিশন মেথডের উপর ভিত্তি করে লেখা।

উইকিপিডিয়া থেকে ট্রায়াল ডিভিশন মেথডটা একটু ব্যাখ্যা করা যাক। ব্যাখ্যার জন্য ১০০ সংখ্যাটিকে বিবেচনা করা যাক। ২, ৪, ৫, ১০, ২০, ২৫, ৫০ হল ১০০-এর ভাজকসমূহ। এখানে একটা লক্ষণীয় ব্যাপার রয়েছে। ১০০-এর সর্বোচ্চ ভাজক ১০০/২=৫০। একটি সংখ্যার সর্বোচ্চ ভাজক ঐ সংখ্যার অর্ধেকের সমান বা ছোট - এটা যেকোন সংখ্যার ক্ষেত্রেই সত্য। যাহক, এবার এই ভাজকদের প্রতি আরো শার্লকীয় (শার্লক হোমস) দৃষ্টি দেয়া যাক।

১০০ = ২ × ৫০ = ৪ × ২৫ = ৫ × ২০ = ১০ × ১০ = ২০ × ৫ = ২৫ × ৪ = ৫০ × ২

কি দেখতে পাচ্ছি? ১০-এর পর থেকে হিস্টরি রিপিট হওয়া শুরু করেছে। সেজন্য ১০-এর পরবর্তী ভাজকদেরকে আমরা বিবেচ্য তালিকা থেকে বাদ দিতে পারি। একটা মজার ব্যাপার খেয়াল করেছি? ১০ কিন্তু ১০০-এর বর্গমূল। আসলে যেকোন সংখ্যার মৌলিকতা পরীক্ষায় ঐ সংখ্যার বর্গমূলের চেয়ে বড় ভাজকদেরকে আমরা বিবেচ্য তালিকা থেকে বাদ দিতে পারি। আবার ২ দিয়ে বিভাজ্য সব সংখ্যাকেও কিন্তু বাদ দিতে পারি। তাহলে দেখা যাবে, তালিকা খুবই ছোট হয়ে যাবে এবং অপেক্ষাকৃত কম সময়ে প্রাইমালিটি টেস্ট করা যাবে।

ট্রায়াল ডিভিশন মেথডটা যদি বুঝে থাকি তবে is\_prime() ফাংশনটাও সহজে বুঝতে পারব আমরা। এই ফাংশনের প্যারামিটার n হল একটি পূর্ণসংখ্যা যার মৌলিকতা পরীক্ষা করব আমরা। সংখ্যাটি মৌলিক হলে ফাংশন True রিটার্ন করবে আর না হলে False রিটার্ন করব। প্রথমেই চেক করা যাক, সংখ্যাটা ১ এর সমান বা ছোট কিনা। কারণ, n কে অবশ্যই ১ এর চেয়ে বড় হতে হবে (সংজ্ঞা দ্রষ্টব্য)। প্রথম শর্ত পূরণ না হলে চেক করে দেখা যাক, সংখ্যাটি ৩ এর সমান বা ছোট কিনা। সংখ্যাটি ৩ এর সমান (মানে ৩) বা ৩ এর চেয়ে ছোট (মানে ২) হলে তা অবশ্যই মৌলিক সংখ্যা হবে। দ্বিতীয় শর্ত পূরণ না হলে চেক করে দেখা যাক, সংখ্যাটি ২ অথবা ৩ দিয়ে নিংশেষে বিভাজ্য কিনা। কারণ, নিংশেষে বিভাজ্য হলে সংখ্যাটি অবশ্যই মৌলিক হবে না।

তিনটি শর্তের কোনটাই পূরণ না হলে আমরা ট্রায়াল ডিভিশন মেথড প্রয়োগ করেছি। সেক্ষেত্রে ৫ থেকে শুরু করে n এর বর্গমূলের সমান অবধি বিবেচ্য প্রতিটি ভাজক দিয়ে ভাগ করে দেখব n নিংশেষে বিভাজ্য হয় কিনা। এজন্য আমরা ভাজকের আইডেন্টিকাল ভ্যারিয়েবল i-এ ৫ অ্যাসাইন করেছি। আমাদের ভাজক যেহেতু সবসময় n এর বর্গমূলের সমান বা ছোট হবে, তাই while ব্লকে প্রতিবার চেক করে দেখব যে i এর বর্গ n এর সমান বা ছোট কিনা। (হয়ত ব্যাপারটা কঠিন হয়ে যাচ্ছে। কিন্তু ছোটবেলায় গণিতের ক্লাসে ফাঁকি না দিয়ে থাকলে বুঝতে কষ্ট হবার কথা নয়।) এবার ভাজক i দিয়ে ভাজ্য n কে ভাগ করে ভাগশেষ শূন্য কিনা চেক করে দেখার পালা। এখানে একটা চ্যালেঞ্জ রয়েছে সবার জন্য। এই if ব্লকটা নিজে নিজে ব্যাখ্যা করার চেষ্টা করতে হবে। এই ব্লকটা কতটুকু শুদ্ধ, শুদ্ধ না হলে সমস্যা কোথায় - এসব খুঁজে বের করতে হবে।

## ইটারেটর ও জেনারেটর

#### ইটারেটর

লুপ চাপ্টারে for লুপের ভিতরে আমরা ইটারেটরের হালকা-পাতলা প্রয়োগ দেখেছিলাম। আমরা দেখেছিলাম, for লুপ দিয়ে লিস্ট, টাপল, ডিকশনারি, স্ট্রিং এসব অবজেক্টকে ইটারেট করা যায়। ইটারেট করা যায় বলে এদেরকে ইটারেবল অবজেক্ট বলে। বিল্ট-ইন ফাংশন iter() এর ভিতর যদি আমরা কেন ইটারেবল অবজেক্টকে পাস করি তবে তা একটা ইটারেটর রিটার্ন করে। আর \_\_next\_\_() মেথড দিয়ে আমরা একটা ইটারেটরের পরবর্তী এলিমেন্ট বা আইটেমকে অ্যাক্সেস করতে পারি। সবগুলো আইটেম অ্যাক্সেস করতে করতে যখন আর কোন আইটেম বাকি থাকে না তখন পাইথন StopIteration এরর থ্রো করে। একটা উদাহরণ দেখা যাক।

>>> x = iter([1, 2, 3])

>>> x

<list\_iterator object at 0x7f0f1694b1d0>

>>> x.\_\_next\_\_()

1

>>> x.\_\_next\_\_()

2

>>> x.\_\_next\_\_()

3

>>> x.\_\_next\_\_()

Traceback (most recent call last):

File "<pyshell#6>", line 1, in <module>

x.\_\_next\_\_()

StopIteration

আমরা এখন একটা ইটারেটর ক্লাস পয়দা করব। আমাদের এই ইটারেটর বিল্ট-ইন range() ফাংশনের মত কাজ করবে, তবে রিভার্সলি। মানে range(5) মানে তো 0, 1, 2, 3, 4। কিন্তু আমাদের তৈরি ইটারেটর ক্লাস এটাকে 5, 4, 3, 2, 1, 0 এরকম কিছু একটা বানাবে। তো চেষ্টা করা যাক।

class revrange:

def \_\_init\_\_(self, n):

self.n = n

self.i = n

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

if self.n >= 0:

if self.i == self.n:

self.n = self.n - 1

return self.i

else:

self.i = self.n

self.n = self.n - 1

return self.i

else:

raise StopIteration

for temp in revrange(5):

print(temp)

**আউটপুট**

5

4

3

2

1

0

\_\_iter\_\_() মেথডটা কিন্তু খুব গুরুত্বপূর্ণ এখানে। এটা ছাড়া পুরো ক্লাসটাই অচল হয়ে থাকবে।

#### জেনারেটর

জেনারেটর একটা ফাংশন। এই ফাংশনের কাজ হল yield স্টেটমেন্ট ব্যবহার করে সিকুয়েন্স পয়দা করা। এইদিক থেকে জেনারেটরও এক ধরনের ইটারেটর।

def revrange(n):

while n >= 0:

yield n

n = n - 1

for temp in revrange(5):

print(temp)

**আউটপুট**

5

4

3

2

1

0

যখন একটা জেনারেটর ফাংশন কল করা হয় তখন এটা ফাংশনের কোড এক্সিকিউট হবার আগেই একটা জেনারেটর অবজেক্ট রিটার্ন করে। যখন প্রথমবার \_\_next\_\_() মেথড কল করা হয় তখন ফাংশনের কোড এক্সিকিউট হওয়া শুরু করে। এক্সিকিউট হতে হতে যখন yield স্টেটমেন্ট অবধি পোঁছায় তখন \_\_next\_\_() মেথড ইয়েল্ডেড ভ্যালু রিটার্ন করে।

## ফাংশন

Published by [subeen](http://pybook.subeen.com/author/subeen/) on [April 16, 2018](http://pybook.subeen.com/2018/04/)

ফাংশনের সঙ্গে পরিচয় আমাদের অনেক আগেই হয়ে গিয়েছে। প্রথমেই যখন Hello world প্রিন্ট করেছিলাম, তখনই আমরা print() ফাংশন ব্যবহার করেছি। এছাড়াও আরো অনেক ফাংশন ব্যবহার করেছি, যেমন input(), len(), type() ইত্যাদি। আবার আমরা turtle মডিউল যখন ব্যবহার করেছি, তখন টার্টলের অনেক ফাংশনও ব্যবহার করেছি, যেমন forward(), left(), right(), dot() ইত্যাদি। ফাংশনগুলো কী কাজ করে, এটি আমাদের জানতে হবে, কিন্তু কিভাবে কাজ করে, সেটি এখন আমাদের জানার প্রয়োজন নেই। এসব ফাংশন তৈরি করে দেওয়া না থাকলে আমাদেরকে অনেক বেশি কষ্ট করতে হতো। যেমন print() ফাংশনটি কিভাবে কাজ করবে, সেটি পাইথনের বিল্টইনস্ মডিউলে লেখা আছে (তবে এই মডিউলটি আমাদের ইমপোর্ট করতে হয় না)। তেমনি dot() কিভাবে কাজ করবে, সেটি লেখা আছে টার্টল মডিউলের ভেতরে। এই অধ্যায়ে আমরা দেখবো, কিভাবে ফাংশন তৈরি করতে হয়, কিভাবে মডিউল তৈরি করতে হয় এবং প্রয়োজনীয় কিছু ফাংশনের ব্যবহার।

আমরা কখন ফাংশন তৈরি করবো? যখন আমাদেরকে একটি নির্দিষ্ট কাজ করতে হবে এবং সেই কাজটি একাধিকবার করার প্রয়োজন হবে, তখন আমরা সেই কাজের জন্য একটি ফাংশন তৈরি করে ফেলবো। আবার কখনও কখনও একটি কাজ একবার করলেও আমরা সেটির জন্য আলাদা ফাংশন তৈরি করতে পারি, যেন কোড বুঝতে সহজ হয়। যেমন ধরা যাক, আমাকে কোনো প্রোগ্রামে অনেকবার দুটি সংখ্যা যোগ করতে হবে। তখন আমরা এই যোগ করার জন্য একটি ফাংশন তৈরি করে ফেলতে পারি এভাবে :

def add(n1, n2):

return n1 + n2

আমরা দেখতে পাচ্ছি যে, ফাংশনের শুরুতে লিখতে হবে def, তাহলে পাইথন বুঝবে যে এখানে একটি ফাংশন তৈরি করা হচ্ছে (বা সংজ্ঞায়িত করা হচ্ছে, definition শব্দের প্রথম তিন অক্ষর def)। তারপর ফাংশনের নাম দিতে হবে। আমরা নাম দিয়েছি add। ফাংশনের নাম দেখে যেন বোঝা যায় যে, ফাংশনটি কী কাজ করবে। তারপরে প্রথম বন্ধনীর ভেতরে ফাংশনের প্যারামিটার লিখতে হবে। সব ফাংশনে প্যারামিটার থাকে না। প্যারামিটার থাকবে কী না এবং কয়টি, সেটি নির্ভর করে আমরা কী ফাংশন তৈরি করছি, তার ওপর। যেমন এখানে আমি তৈরি করছি দুটি সংখ্যার যোগফল বের করার ফাংশন। তাহলে তো আমাকে কোন দুটি সংখ্যা আমি যোগ করবো, সেগুলো জানতে হবে বা ইনপুট নিতে হবে। ফাংশনের ক্ষেত্রে প্যারামিটার হচ্ছে ইনপুট নেওয়ার পদ্ধতি। তারপরে একটি কোলন চিহ্ন দিতে হবে। পরের লাইন থেকে ফাংশনের ভেতরের কোড লিখতে হবে এবং সেগুলো ইনডেনটেশন করা থাকতে হবে, নইলে পাইথনের পক্ষে বোঝা সম্ভব হবে না যে কোন অংশ ফাংশনের ভেতরে আর কোন অংশ বাইরে। ফাংশন থেকে আবার এক বা একাধিক জিনিস ফেরত পাঠানো যায়, যাকে বলে রিটার্ন করা। আমাদের যেমন যোগফল ফেরত পাঠানো দরকার, তাই আমরা n1 + n2 এর মান রিটার্ন করছি। আমরা এখন ফাংশনটির ব্যবহার দেখবো। আমাদের কোনো নিয়মকানুন মুখস্থ করার প্রয়োজন নেই, চর্চা করে ও চিন্তা করে আমরা প্রোগ্রামিং শিখবো।

>>> def add(n1, n2):

... return n1 + n2

...

>>> n = 10

>>> m = 5

>>> result = add(n, m)

>>> print(result)

15

>>>

>>> number1 = 10

>>> number2 = 10

>>> result = add(number1, number2)

>>> print(result)

20

>>>

>>> n1 = 20

>>> n2 = 10

>>> print(add(n1, n2))

30

>>> print(add(2.5, 3.5))

6.0

>>>

আমরা ফাংশনটির বিভিন্ন রকম ব্যবহার দেখলাম। যেসব বিষয় খেয়াল করতে হবে, সেগুলো হচ্ছে –

* যদিও ফাংশনের প্যারামিটার হিসেবে n1 ও n2 ব্যবহার করা হয়েছে, আমরা কিন্তু ফাংশন কল করার সময় যেকোনো নামের আর্গুমেন্ট ব্যবহার করতে পারি।
* আমরা যখন ফাংশনটি সংজ্ঞায়িত বা ডিফাইন করছি বা সহজ বাংলায় ফাংশনটি তৈরি করছি, তখন যে n1, n2 লিখলাম, এগুলোকে বলে ফাংশনের প্যারামিটার (parameter)। আর আমরা যখন ফাংশনটি কল করছি, যেমন add(n, m) বা add(n1, n2) বা add(2, 3) এখানে n, m বা n1, n2 বা 2, 3 হচ্ছে আর্গুমেন্ট (argument)। নামগুলো মুখস্থ করতে হবে না, তবে জেনে রাখা ভালো। ভবিষ্যতে ইংরেজি বই পড়ার সময় কাজে লাগবে।
* আর্গুমেন্ট হিসেবে ভ্যারিয়েবল ব্যবহার করা যায়, আবার সরাসরি বিভিন্ন সংখ্যাও ব্যবহার করা যায়।

ফাংশনের যে সবসময় প্যারামিটার থাকতেই হবে, সেরকম কোনো কথা নেই। আবার ফাংশন থেকে যে কোনো কিছু রিটার্ন করতে হবে, এমন কোনো কথা নেই। আমরা ফাংশন থেকে কোনো কিছু রিটার্ন না করলে পাইথন আপনাআপনি None রিটার্ন করে। এটি পরীক্ষা করে দেখা যেতে পারে। আবার অনেকসময় ফাংশন থেকে একাধিক জিনিসও রিটার্ন করতে হতে পারে। সেক্ষেত্রে যেসব জিনিস রিটার্ন করতে হবে, return স্টেটমেন্টের পর সেগুলো কমা দিয়ে পৃথক করে দিতে হবে। যেমন : return a, b, c।

আমরা টার্টল ব্যবহার করে বর্গক্ষেত্র আঁকার একটি প্রোগ্রাম তৈরি করেছিলাম। প্রোগ্রামটিতে আমরা বর্গক্ষেত্র আঁকার কাজটি একটি ফাংশন তৈরি করে করতে পারি।

def draw\_square(side\_length):

for i in range(4):

turtle.forward(side\_length)

turtle.left(90)

তাহলে একটি সম্পূর্ণ প্রোগ্রাম লিখে দেখি, সেই বৃত্ত তৈরির প্রোগ্রামটি। নিচের প্রোগ্রামটি সাদাকালো, তবে কেউ চাইলে বিভিন্ন রং ব্যবহার করতে পারে :

import turtle

def draw\_square(side\_length):

for i in range(4):

turtle.forward(side\_length)

turtle.left(90)

counter = 0

while counter < 90:

draw\_square(100)

turtle.right(4)

counter += 1

turtle.exitonclick()

অনুশীলনী: একটি ফাংশন তৈরি করতে হবে যা প্যারামিটার হিসেবে একটি বাহুর দৈর্ঘ্য নেবে এবং একটি সমবাহু ত্রিভূজ আঁকবে।

এখন আমরা কিছু উদাহরণের মাধ্যমে ফাংশন সম্পর্কে আরো জানবো।

def myfnc(x):

print("inside myfnc", x)

x = 10

print("inside myfnc", x)

x = 20

myfnc(x)

print(x)

ওপরের প্রোগ্রামটি রান করলে আমরা আউটপুট পাবো এমন :

inside myfnc 20

inside myfnc 10

20

তাহলে আমরা দেখতে পেলাম, ভ্যারিয়েবলের নাম যদিও একই, কিন্তু myfnc-এর ভেতরে x-এর মান পরিবর্তন করে দিলেও ফাংশনের বাইরে x-এর মান পরিবর্তিত হয় নি। এর কারণ হচ্ছে myfnc যখন কল হয়, তখন সে আর্গুমেন্ট হিসেবে যেসব ভ্যারিয়েবল পায়, সেগুলোর একটা কপি তৈরি হয়। তাই myfnc-এর x আর তার বাইরের x আসলে দুটি আলাদা ভ্যারিয়েবল। একটি ফাংশনের ভেতরে যেসব ভ্যারিয়েবল তৈরি করা হয় সেগুলো হচ্ছে ওই ফাংশনের লোকাল (local) ভ্যারিয়েবল। ফাংশনের বাইরে তার অস্তিত্ব থাকে না।

ফাংশনের বাইরে যদি কোনো ভ্যারিয়েবল থাকে, তাহলে ফাংশনের ভেতর থেকে ওই ভ্যারিয়েবল পাওয়া যায়। একে বলে গ্লোবাল (global) ভ্যারিয়েবল। নিচের প্রোগ্রাম রান করলে সেটি আমরা দেখতে পাবো :

def myfnc(y):

print("y =", y)

print("x =", x)

x = 20

myfnc(x)

প্রোগ্রামের আউটপুট হবে এরকম :

y = 20

x = 20

এখন আমরা যদি myfnc এর বাইরে থেকে y-এর মান দেখতে চাই, সেটি কি সম্ভব? নিচের প্রোগ্রাম রান করলেই বুঝতে পারা যাবে।

def myfnc(y):

print("y =", y)

print("x =", x)

x = 20

myfnc(x)

print("y:", y)

আউটপুট হবে এরকম :

y = 20

x = 20

Traceback (most recent call last):

File "fnc\_test.py", line 7, in <module>

print("y:", y)

NameError: name 'y' is not defined

অর্থাৎ y নামে কোনো কিছু পাওয়া যায় নি।

এখন আমরা দেখবো, ফাংশনের প্যারামিটারের ডিফল্ট (default) মান কিভাবে দেওয়া যায়।

def myfnc(y=10):

print("y =", y)

x = 20

myfnc(x)

myfnc()

এই প্রোগ্রামে আমি দুইবার myfnc ফাংশন কল করলাম। প্রথমবার আর্গুমেন্ট হিসেবে x পাঠাচ্ছি। দ্বিতীয়বার কিছুই পাঠাচ্ছি না। কিন্তু ফাংশনের প্যারামিটারে আবার বলে দিয়েছি y=10। এর মানে হচ্ছে যদি কোনো আর্গুমেন্ট পাঠানো না হয়, তাহলে y-এর মান হবে 10, আর যদি কোনো আর্গুমেন্ট পাঠানো হয়, তাহলে আর্গুমেন্টে যেই ভ্যারিয়েবলটি পাঠানো হলো, সেই ভ্যারিয়েবলের মান y-তে কপি হবে।

এবারে আরেকটি প্রোগ্রাম লিখবো :

def myfnc(x, y=10, z):

print("x =", x, "y =", y, "z =", z)

x = 5

y = 6

z = 7

myfnc(x, y, z)

ওপরের কোড যদি রান করি, তাহলে আউটপুট পাবো এরকম :

def myfnc(x, y=10, z):

^

SyntaxError: non-default argument follows default argument

এই এররের অর্থ হচ্ছে আমরা যদি ডিফল্ট মান দেই, তাহলে তারপরে সবগুলো আর্গুমেন্টে ডিফল্ট মান থাকতে হবে। মানে আমরা যদি লিখতাম def myfnc(x, y=10, z=0) তাহলে এই এরর আর আসবে না। আমরা প্রোগ্রাম লিখে পরীক্ষা করে দেখি। নিচের প্রোগ্রামটি রান করে আউটপুট দেখে বুঝে নিতে হবে।

def myfnc(x, y=10, z=0):

print("x =", x, "y =", y, "z =", z)

x = 5

y = 6

z = 7

myfnc(x, y, z)

myfnc(x, y)

myfnc(x)

আর আমরা যদি চাই যে, z-এর কোনো ডিফল্ট মান থাকবে না তাহলে এভাবে লিখতে হবে : myfnc(x, z, y = 10)। কিন্তু এভাবে লিখলে তো একটা সমস্যা আছে, দ্বিতীয় আর্গুমেন্ট যেটি পাঠাবো, সেটি তো z-এ আসবে, কারণ প্যারামিটারের নাম তো বিবেচনা করা হয় না, বরং প্রথমটি x-এ, দ্বিতীয়টি z-এ এবং তৃতীয়টি y-এ কপি হবে। কিন্তু পাইথনে একটি উপায় আছে যেখানে নির্দিষ্ট প্যারামিটারে নির্দিষ্ট মান পাঠানো যায়। নিচের উদাহরণে আমরা সেটি দেখবো।

def myfnc(x, z, y=10):

print("x =", x, "y =", y, "z =", z)

myfnc(x = 1, y = 2, z = 5)

a = 5

b = 6

myfnc(x = a, z = b)

a = 1

b = 2

c = 3

myfnc(y = a, z = b, x = c)

প্রোগ্রামটি রান করলে নিচের মতো আউটপুট আসবে :

x = 1 y = 2 z = 5

x = 5 y = 10 z = 6

x = 3 y = 1 z = 2

পাইথনে সুযোগ-সুবিধা অনেক বেশি। তবে এখানে একটি কথা বলা প্রয়োজন। বইটি প্রথমবার পড়ার পরে বেশিরভাগ পাঠকেরই এত কিছু মনে থাকবে না এবং এটি খুবই স্বাভাবিক। এতে চিন্তিত হওয়ার কিছু নেই। বইটি দ্বিতীয় বা তৃতীয়বার পড়ার (এবং সেই সঙ্গে প্রতিবারই বইয়ের সব উদাহরণ ও অনুশীলনীগুলো করতে হবে) পর এগুলো মনে থাকবে।

আমরা চাইলে ফাংশনের ভেতরে লিস্টও পাঠাতে পারি। যেমন, এখন আমি একটি ফাংশন লিখবো, যেটিতে আর্গুমেন্ট হিসেবে একটি লিস্ট পাঠানো যাবে এবং আমরা সেই লিস্টের সংখ্যাগুলো যোগ করে যোগফল ফেরত পাঠাবো। যদিও লিস্টে আমরা যেকোনো কিছুই পাঠাতে পারি; কিন্তু যেহেতু যোগ করবো, তাই আমরা সংখ্যার লিস্ট পাঠাবো।

def add\_numbers(numbers):

result = 0

for number in numbers:

result += number

return result

result = add\_numbers([1, 2, 30, 4, 5, 9])

print(result)

এখন, ফাংশনের ভেতর ভ্যারিয়েবল পাঠানো আর লিস্ট পাঠানোর মধ্যে একটি পার্থক্য আছে। যে ফাংশনের ভেতরে লিস্ট পাঠানো হয়েছে, সেখানে যদি লিস্টটি পরিবর্তন করা হয়, তাহলে আসল লিস্টও পরিবর্তিত হয়ে যায়। এটিও আমরা একটি প্রোগ্রাম লিখে যাচাই করো নেবো। তবে তার আগে আমাদের আরো একটি জিনিস জেনে নিতে হবে। আমরা যদি লিস্টের কোনো নির্দিষ্ট উপাদান পেতে চাই, যেমন প্রথম, দ্বিতীয়, তৃতীয় ইত্যাদি, তাহলে আমরা তৃতীয় বন্ধনীর ভেতরে সেই সংখ্যাটি উল্লেখ করে দিতে পারি। একে বলা হয় ইনডেক্স। তবে পাইথনে লিস্টের ইনডেক্স কিন্তু 1 থেকে নয়, 0 থেকে শুরু হয়। অর্থাৎ, আমরা যদি লিস্টের প্রথম উপাদান একসেস করতে চাই, তাহলে আমরা লিস্টের নামের পর [0] লিখবো। country নামের একটি লিস্টে যদি পৃথিবীর সবগুলো দেশের নাম থাকে, আর সেই লিস্টের 50-তম দেশটি আমরা দেখতে চাই, তাহলে আমাদেরকে country[49] প্রিন্ট করতে হবে।

def test\_fnc(li):

li[0] = 10

my\_list = [1, 2, 3, 4]

print("before function call", my\_list)

test\_fnc(my\_list)

print("after function call", my\_list)

প্রোগ্রামটি রান করলে আমরা আউটপুট পাবো এরকম :

before function call [1, 2, 3, 4]

after function call [10, 2, 3, 4]

এরকম হওয়ার কারণ কী? কারণ আমরা যখন একটি লিস্ট অন্য একটি ভ্যারিয়েবলে অ্যাসাইন করি, তাহলে নতুন কোনো লিস্ট তৈরি হয় না, বরং নতুন ভ্যারিয়েবল আর পুরনো ভ্যারিয়েবলে একই লিস্ট থাকে। সেটিও আমরা পরীক্ষা করে দেখবো পাইথন ইন্টারপ্রেটারে।

>>> list1 = [1, 2, 3, 4]

>>> list2 = list1

>>> print(list1)

[1, 2, 3, 4]

>>> print(list2)

[1, 2, 3, 4]

>>> list2[0] = 100

>>> print(list2)

[100, 2, 3, 4]

>>> print(list1)

[100, 2, 3, 4]

তাই ফাংশনের ভেতরে লিস্ট পাঠানোর সময় কিংবা আরেকটি ভ্যারিয়েবলে লিস্ট অ্যাসাইন করার সময় আমাদের সতর্ক হতে হবে। নইলে প্রোগ্রামে বাগ-এর সৃষ্টি হবে। কম্পিউটার প্রোগ্রামের ত্রুটিকে বলে বাগ (bug)।

আমরা যে একটু আগে লিস্টের উপাদানগুলো যোগ করার জন্য একটি ফাংশন লিখলাম, পাইথনে কিন্তু সেই কাজ করার জন্য একটি বিল্টইন ফাংশন আছে। আমাদের প্রয়োজন হলে আমরা সেই ফাংশন ব্যবহার করবো। তাহলে আমি উদাহরণের প্রোগ্রাম কেন লিখলাম? ফাংশনে কিভাবে লিস্ট পাঠাতে হয়, সেটি দেখানোর জন্য।

>>> li = [1, 2, 3]

>>> result = sum(li)

>>> print(result)

6

আমরা বইয়ের পরের অধ্যায়গুলোতে আরো বিভিন্ন বিল্টইন ফাংশন ব্যবহার করবো এবং নিজেরাও কিছু কিছু ফাংশন তৈরি করবো।

## পাইথন প্রোগ্রামিং এ ফাংশন

এর আগে আমরা বিভিন্ন ফাংশন যেমন print, input ইত্যাদি ফাংশন ব্যবহার করেছি। এগুলো প্রোগ্রামিং ল্যাঙ্গুয়েজের এর সাথে দিয়ে দেওয়া হয়েছে যেন আমরা সহজেই প্রোগ্রাম লিখতে পারি। এ অধ্যায় শিখব কিভাবে নিজের প্রয়োজন মত ফাংশন লিখে ফেলা যায়।  
ফাংশন হচ্ছে পুনরায় ব্যবহার যোগ্য কোড ব্লক। যা একটি নির্দিষ্ট কাজ করতে পারে। ফাংশন ভালো ভাবে জানলেই প্রোগ্রামিং এর একটা বিশাল অংশ শেখা শেষ হয়ে যায়। লেখা যায় নিজের ইচ্ছে মত কোড।

**ফাংশন লেখার নিয়মঃ**

একটা ফাংশন নিচের মত করে লেখা হয়ঃ

|  |  |
| --- | --- |
| 1  2  3 | def function-name(Parameters):      statements      return |

এখানে  
• def দিয়ে ফাংশন শুরু করা হয়।  
• ফাংশনের তো একটা নাম থাকতে হবে তাই না? যে নাম দিয়ে ফাংশনটিকে ডাকতে হবে। function-name হচ্ছে ফাংশনের নাম।  
• Parameters হচ্ছে ফাংশন দিয়ে কি কি ডেটা পাস করবে। এখানে এক বা একাদিক Parameter পাস করা যেতে পারে। কোন কোন ফাংশনে কোন Parameter নাও থাকতে পারে। এটা নির্ভর করে কি ধরনের ফাংশন লিখা হচ্ছে তার উপর। একের অধিক Parameter থাকলে তাদেরকে কমা দিয়ে লিখতে হয়।  
• কাজ শেষে ফাংশনটি কি রিটার্ন করবে তাই return দিয়ে পাস করা হয়। ফাংশন যদি কোন কিছু রিটার্ন না করে, তাহলে return 0 বা না লিখলেও হবে।  
যেমন আমরা hello নামে একটা ফাংশন লিখব, যেটাকে কল করলে Hello World! প্রিন্ট করবে।

|  |  |
| --- | --- |
| 1  2 | def hello():      print ("Hello world!") |

উপরের ফাংশনে কোন প্যারামিটার নেই, এবং ফাংশনটি কোন কিছু রিটার্ণ ও করবে না। শুধু Hello World! প্রিন্ট করবে। ফাংশন লেখার পর তা ব্যবহার করার জন্য কল করতে হয়।

|  |  |
| --- | --- |
| 1 | hello() |

কল করলে ফাংশনটি এক্সিকিউট হবে। সম্পুর্ণ প্রোগ্রামঃ

|  |  |
| --- | --- |
| 1  2  3  4 | def hello():      print ("Hello world!")    hello() |

আমরা যত বার ইচ্ছে ততবার ফাংশটি কল করতে পারি। এটাই হচ্ছে ফাংশনের মূল সুবিধে। কোড গুলো বার বার না লেখে শুধু আমরা কল করব। আর তখন ফাংশনটি এক্সিকিউট হয়ে আমরা যা চাই, তা করে দিবেঃ

|  |  |
| --- | --- |
| 1  2  3  4  5 | def hello():   print ("Hello world!")  hello()  hello()  hello() |

প্রোগ্রামটি রান করলে তিনবার Hello World! প্রিন্ট করবে।

**ফাংশনের প্যারামিটার**

আমাদের এমন ফাংশন লিখতে হতে পারে, যেখানে ফাংশনে কিছু ডেটা পাঠালে ঐ ডেটা আমাদের প্রসেস করে দিবে। আর এই ডেটা আমরা পাঠাবো প্যারামিটার হিসেবে।  
আমরা add নামে একটা ফাংশন লিখব। যেটা প্যারামিটার হিসেবে দুইটা নাম্বার নিবে এবং রিটার্ন করবে নাম্বার দুটির যোগফলঃ

|  |  |
| --- | --- |
| 1  2  3 | def add(a,b):      return (a+b)  print (add(4,7)) |

এখানে add ফাংশনে আমরা দুইটা সংখ্যা প্যারামিটার হিসেবে দিচ্ছি। ফাংশনটি আমাদের ফেরত দিচ্ছে সংখ্যা দুইটির যোগফল।

যখন আমরা ফাংশনটি কল করেছি, তখন সংখ্যা দুটি পাস করে দিয়েছি, যে সংখ্যা দুইটির যোগ ফল আমরা পেতে চাই, add(4,7) দিয়ে। ফাংশনটি আমাদের ঐ সংখ্যা দুটি নিয়ে তা যোগ করে তা ফেরত দিয়েছে, return (a+b) দিয়ে। পরে তা প্রিন্ট করেছি।

এখন আমরা যতবার ইচ্ছে ততবার add ফাংশনটি কল করতে পারি। যাতে দুইটি সংখ্যা প্যারামিটার হিসেবে দিলে তা যোগ করে আমাদের পাঠিয়ে দিবে। যেন আমরা তা প্রিন্ট করতে পারি। আলাধা করে যোগ করতে হচ্ছে না।

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | def add(a,b):      return (a+b)    print (add(4,7))  print (add(89,732))  print (add(55,999)) |

যা আমাদের আউটপুট দিবেঃ  
11  
821  
1054  
**ডিফল্ট প্যারামিটারঃ**   
কোন ফাংশন লেখার সময় আমরা তার ডিফল্ট প্যারামিটার সেট করে দিতে পারি। ডিফল্ট প্যারামিটার সেট করে দেওয়া হলে যদি ফাংশন কল করার সময় ঐ প্যারামিটারের আইটেমের জন্য কোন ভ্যালু সেট করে দেওয়া না হয়, তাহলে ফাংশন ডিফল্ট ভ্যালুটি ব্যবহার করবে।

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | def max(a,b=0):        if(a>b):          return format(a) +" is greater than " + format(b)      elif (a<b):          return format(a) + " is less than " + format(b)      elif(a==b):          return format(a) + " and " + format(b) + " both are same"    print(max(6,7))  print(max(9))  print(max(5,5)) |

যা আউটপুট দিবেঃ  
6 is less than 7  
9 is greater than 0  
5 and 5 both are same  
উপরের প্রোগ্রামটি দেখি। আমরা দুইটা সংখ্যার মধ্যে বড় ছোট নির্ণয়ের জন্য একটা ফাংশন লিখেছি। যখন ফাংশনে দ্বিতীয় প্যারামিটার হিসেবে কোন ভ্যালু পাঠাই নি, তখন ডিফল্ট ভ্যালু 0 ব্যবহার করেছে।

**ফাংশন থেকে একাধিক ভ্যালু রিটার্ন**   
এর আগের প্রোগ্রামে দেখেছি ফাংশন থেকে কিভাবে একটা ভ্যালু রিটার্ন করা যায়। পাইথনে ফাংশন থেকে একাধিক ভ্যালু রিটার্ন করা যায়। নিচের প্রোগ্রামটি দেখিঃ

|  |  |
| --- | --- |
| 1  2  3  4 | def binary(a, b):      return bin(a), bin(b)    print(binary(12, 14)) |

এখানে একটি ফাংশন লিখেছি, যেখানে দুইটা সংখ্যা পাঠালে ঐ সংখ্যা দুইটির বাইনারি ভ্যালু রিটার্ণ করবে। যদিও দুইটা একসাথে টাপল আকারে প্রিন্ট করবে। আমরা চাইলে প্রতিটা রিটার্ণ ভ্যালু আলাদা আলাদা এক্সেস করতে পারিঃ

|  |  |
| --- | --- |
| 1  2  3  4 | def binary(a, b):      return bin(a), bin(b)  x,y = (binary(12, 14))  print(x) |

এখানে রিটার্ণ ভ্যালু গুলো x এবং y তে আলাদা আলাদা ভাবে এসাইন হবে। আমরা এরপর যে কোন ভ্যালু প্রিন্ট করে দেখতে পারি।

আরো দুই একটা ফাংশন দেখিঃ

আমরা একটা ফাংশন লিখব। যেটা ব্যবহারকারী কে তার নাম জিজ্ঞেস করবে। এবং তা ফাংশনে পাস করব। ফাংশন পরে ব্যবহার কারীকে হ্যালো জানাবেঃ

|  |  |
| --- | --- |
| 1  2  3  4  5 | def hello(name):      print ("Hello " + name)    your\_name = input("Enter  your Name: ")  hello(your\_name) |

উপরের প্রোগ্রামে আমরা hello নামে একটা ফাংশন লিখেছি। ব্যবহারকারী থেকে আমরা তার নাম ইনপুট নিয়েছি। এরপর hello ফাংশনে তা পাস করেছি, hello(your\_name) দিয়ে। hello ফাংশন আমাদের দেওয়া নামকে হ্যালো জানিয়েছে।  
এবার আমরা আরেকটা ফাংশন লিখব। যেটা একটি নাম্বার জোড় না বিজোড় তা বলে দিবে। ব্যবহারকারী থেকে একটা নাম্বার ইনপুট নিব। তারপর তা ফাংশনে পাস করব। ফাংশন আমাদের বলে দিবে নাম্বারটি জোড় না বিজোড়ঃ

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | def checkNumber(n):      if n%2 == 0:          print ("It's Even number")      else: print ("It's Odd number")    number = int (input("Enter  a  number to check: "))  checkNumber(number) |

এভাবে আমরা আমাদের প্রয়োজন মত যে কোন ফাংশন লিখে ফেলতে পারি।

## PYTHON BASIC PART-8: পাইথনে ফাঙ্কশন কি?

### পাইথনে ফাঙ্কশন কি? What is function in Python?

Python এ function হচ্ছে Program এর এমন একটা code block যা একটা নির্দিষ্ট কার্য সম্পাদনের এবং তা বার বার ব্যবহারের জন্যে ব্যবহৃত হয়। অর্থাৎ function এর মধ্যে এমন কিছু Instruction বা নির্দেশাবলী থাকে যা function ব্যবহারকারী থেকে এক বা একাধিক input নিয়ে এক বা একাধিক output বা result রিটার্ন করার জন্য ব্যবহৃত হয়। তবে অনেক সময় ইনপুট ছাড়াও আমরা আউটপুট রিটার্ন করতে পারি।

ফাঙ্কশনের মাধ্যমে প্রোগ্রামের যে অংশগুলো বার বার আসে সেগুলোকে আমরা পুনরায় ব্যবহারযোগ্য করতে পারি ।

পাইথনে ফাংশনকে অনেকটা আমরা একটা ছোট্ট মেশিনের সাথে তুলনা করতে পারি, অর্থাৎ একবার মেশিনটি তৈরি করবো আর বার বার ব্যবহার করে কিছু একটা কাজ করব বা জিনিষ তৈরি করবো। কিছু তৈরি করতে হলে মেশিনে কিছু ইনপুট দিতে হতে পারে। আবার কিভাবে তৈরি করবে সেটাও মেশিনের মধ্যে যন্ত্রপাতি বসিয়ে সেটআপ করতে হবে। এভাবে বাস্তবে একটা মেশিন তৈরি করাকেই প্রোগ্রামের মধ্যে ফাংশন ডিফাইন করা বলা যেতে পারে।

পাইথনে সাধারণত, program page লোড হওয়ার সাথে সাথেই ডিক্লেয়ার করা function execute হয়না, যতক্ষণ না function কে call করা না হয়।

### পাইথনে function কত প্রকার ?

দুই প্রকার :

* Python Pre-built function বা Python language কর্তৃক পূর্ব নির্ধারিত বিভিন্ন library function বা library ছাড়া function
* User/Programmer defined function বা Programmer কর্তৃক নির্ধারিত function

### পাইথনে function কিভাবে লিখতে হয়?

* পাইথনে ফাংশন ডিক্লেয়ার করার জন্য def কি-ওয়ার্ডটি দিয়ে শুরু করতে হবে।
* def কি-ওয়ার্ডটির পর ফাঙ্কশনের নামটি letter অথবা underscore দিয়ে শুরু করতে হয়।
* ফাঙ্কশনের নামের শেষে একজোড়া প্রথম বন্ধনী () এরপর একটি colon (:) দিতে হবে।
* ফাংশনকে কল করার বা ব্যবহার করার আগেই সেই ফাংশনকে প্রোগ্রামে ডিফাইন করতে হবে।
* “””docstring””” অর্থাৎ function এর মধ্যে function এর documentation লেখার জন্য ব্যবহৃত হয়।
* পাইথনে function Case sensitive অর্থাৎ একই নামে কোনো ফাঙ্কশন lowercase এবং uppercase এর কারণে আলাদা function হিসেবে বিবেচিত হবে।

### পাইথনে function লেখার Syntax কি ?

|  |  |
| --- | --- |
| 1  2  3 | def function\_name(parameters):      """docstring"""      statement(s) |

চলুন syntax অনুসারে একটা function তৈরী করে ফেলি :

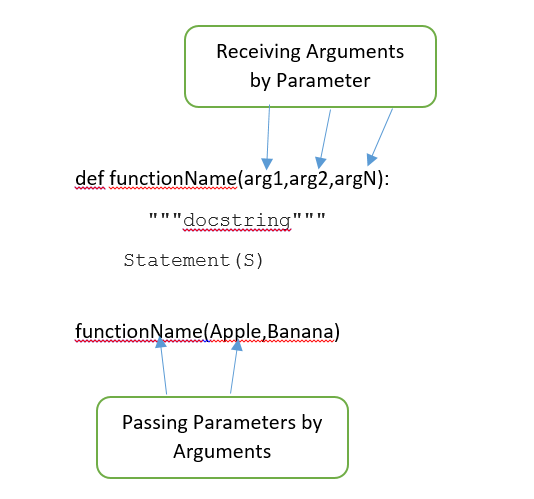
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | def greet(name):      """This function greets to      the person passed in as      parameter"""      print("Hello, " + name + ". Good morning!");    def Greet(name):      """This function Greets to      the person passed in as      parameter"""      print("Hello, " + name + ". Good morning!");    Greet("Masud");  greet("Sohel"); |

### Output

|  |  |
| --- | --- |
| 1  2 | Hello, Masud. Good morning!  Hello, Sohel. Good morning! |

### পাইথনে function এর মধ্যে Parameter কি এবং argument কি ?

function declaration অথবা define এর সময় parentheses এর মধ্যে ব্যবহৃত একেকটি variable কে একেকটি function parameter বলা হয়। একই ভাবে function calling এর সময় parentheses এর মধ্যে ব্যবহৃত একেকটি variable কে একেকটি function argument বলা হয়।function parameter এবং arguments কিভাবে কাজ করে তা বাজার জন্য নিচের ছবিটি লক্ষ্য করুন :



function Parameter and arguments

চলুন function parameter এবং arguments দিয়ে একটা function define এবং call করি।

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | def getSum(num1, num2):    sum = num1 + num2;    print("Sum of the two numbers ",num1," and ",num2,"  is : ", sum);        getSum(10, 20); |

### Output

|  |  |
| --- | --- |
| 1 | Sum of the two numbers  10  and  20   is :  30 |

### পাইথনে functions এর arguments গুলোকে কয় ভাবে receive বা গ্রহণ করা যায়?

পাইথনে দুই ভাবে argument গুলোকে receive বা গ্রহণ করা যায় :

* সরাসরি function এর ভিন্ন ভিন্ন Parameter দিয়ে ভিন্ন ভিন্ন argument গুলো রিসিভ করা যায়।
* function এর একটা নির্দিষ্ট Parameter দিয়ে সবগুলো argument গুলো রিসিভ করা যায়।

এবার চলুন একটা উদাহরণের মাধ্যমে দেখা যাক কিভাবে সরাসরি function এর ভিন্ন ভিন্ন Parameter দিয়ে ভিন্ন ভিন্ন argument গুলো রিসিভ করা যায়:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | def getSum(num1, num2):    sum = num1 + num2;    print("Sum of the two numbers ",num1," and ",num2,"  is : ", sum);        getSum(10, 20); |

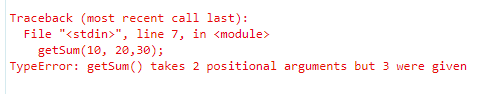
### Output

|  |  |
| --- | --- |
| 1 | Sum of the two numbers  10  and  20   is :  30 |

লক্ষ্য করুন উপরের উদাহরণে আমরা num1 এবং num2 parameter এর জন্য দুটি আলাদা Argument যথাক্রমে 10 এবং 20 পাঠিয়েছি, যদি কম অথবা বেশি পাঠাই তাহলে নিচের মতো error প্রদর্শন করবে:

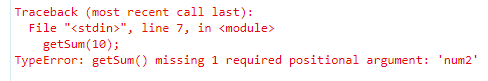
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | def getSum(num1, num2):    sum = num1 + num2;    print("Sum of the two numbers ",num1," and ",num2,"  is : ", sum);        getSum(10, 20,30); |

### Output



python argument missing error

আবার যদি কম পাঠান, তাহলে নিচের মতো error আসবে:



python argument missing error 2

তবে আপনি চাইলে নিচের মতো করে একটা ডিফল্ট ভ্যালু সেট করে দিতে পারেন :

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | def getSum(num1, num2=20):    sum = num1 + num2;    print("Sum of the two numbers ",num1," and ",num2,"  is : ", sum);      getSum(10); |

তবে আপনি চাইলে প্যারামিটার এর নাম ধরেও argument pass করতে পারেন। চলুন একটা উদাহরণে বুঝে নেয়া যাক:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | def getSum(num1, num2):    sum = num1 + num2;    print("Sum of the two numbers ",num1," and ",num2,"  is : ", sum);        getSum(num2=30, num1=20); |

### Output

|  |  |
| --- | --- |
| 1 | Sum of the two numbers  20  and  30   is :  50 |

তবে এক্ষেত্রে সমস্যা হচ্ছে সবগুলো প্যারামিটার এর নাম ধরেই পাঠাতে হয়, আবার সুবিধা হয় আপনি চাইলে প্যারামিটার swaping করতে পারবেন। লক্ষ্য করুন আমরা দ্বিতীয় প্যারামিটার এর ভ্যালু প্রথমে পাঠিয়েছি।

### পাইথনে function এর একটা নির্দিষ্ট Parameter দিয়ে সবগুলো argument গুলো রিসিভ করার নিয়ম :

পাইথনে function এর একটা নির্দিষ্ট Parameter দিয়ে সবগুলো argument গুলো রিসিভ করতে হলে আপনাকে ফাঙ্কশনের parameter এর সামনে একটা “\*” star দিতে হবে। চলুন একটা উদাহরণে বুঝে নেয়া যাক:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | def greet(\*names):     """This function greets all     the person in the names tuple."""       # names is a tuple with arguments     for name in names:         print("Hello",name)    greet("Monir","Minhaz","Iqbal","Sharif") |

### Output

|  |  |
| --- | --- |
| 1  2  3  4 | Hello Monir  Hello Minhaz  Hello Iqbal  Hello Sharif |

### পাইথনে Variable Scope কি?

পাইথনে আপনি আপনার সমস্ত কোড জুড়ে যেকোনো জায়গায় Variable Declare বা ঘোষণা করতে পারেন। কোনো Variable যদি আপনি function এর ভিতরে ঘোষণা করেন, তাহলে ঐ Variable এর ব্যবহার Function এর মধ্যেই সীমাবদ্ধ রাখতে হবে। পাইথনে এইটাকে বলা হয় Local Scope, আবার যদি কোনো Variable ফাঙ্কশনের বাহিরে ঘোষণা করেন। তাহলে ঐ Variable সমস্ত কোড জুড়ে ব্যবহার করতে পারবেন এমনকি Funciton এর ভিতরেও ব্যবহার করতে পারবেন।

পাইথনে Variable Scope দুই ধরনের :

* local
* global

প্রথমে চলুন পাইথনে variable এর global scope এর উদহারণ দেখা যাক:

|  |  |
| --- | --- |
| 1  2  3  4  5 | x = 10  def myFunc():      print("Value inside function:",x)    myFunc() |

### Output

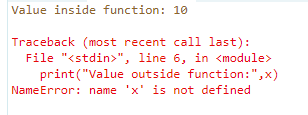
Value inside function: 10

**লক্ষ্য করুন:** আমাদের variable ফাঙ্কশনের বাহিরে ডিক্লেয়ার করা তারপর ও এটা ফাঙ্কশনের ভিতরে ব্যবহার করা গেছে , এবং ফলাফল ও আসছে।

### এবার চলুন পাইথনে variable এর local scope এর উদহারণ দেখা যাক:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | def myFunc():      x = 10      print("Value inside function:",x)    myFunc()  print("Value outside function:",x) |

### Output



python variable localscope

তবে আপনি চাইলে variable কে ফাঙ্কশনের মধ্যে global ডিক্লেয়ার করে , ফাঙ্কশনের বাহিরে variable কে ব্যবহার করতে পারেন। চলুন উদাহরণ দিয়ে বুঝে নেয়া যাক:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | def myFunc():      global x      x=20      print("Value inside function:",x)    myFunc()  print("Value outside function:",x) |

### output

|  |  |
| --- | --- |
| 1  2 | Value inside function: 20  Value outside function: 20 |

### পাইথনে Anonymous এবং lambda function কি?

Anonymous শব্দের অর্থ হচ্ছে নাম বিহীন বা বেনামী, আর anonymous function মানে হচ্ছে নাম বিহীন function বা বেনামী function, সুতরাং পাইথনে যখন কোনো function declare বা ঘোষণা করা হয়, কিন্তু function টির কোনো নাম থাকেনা, পাইথনে সেই function কে anonymous function বলে। আর এই anonymous ফাঙ্কশনকেই আবার lambda function বলে।

সাধারণতঃ পাইথনে function declare করা হয় def কীওয়ার্ড ব্যবহার করে, কিন্তু পাইথনে anonymous ফাংশনগুলি lambda কীওয়ার্ড ব্যবহার করে declare করা হয়।

### পাইথন মধ্যে Lambda/anonymous ফাঙ্কশনের সিনট্যাক্স কি?

|  |  |
| --- | --- |
| 1 | lambda arguments: expression |

এবার চলুন একটা উদাহরণ দিয়ে বুঝে নেয়া যাক:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | # Program to show the use of lambda functions    double = lambda x: x \* 2    # Output: 10  print(double(5)) |

### Output

10

উপরের ফাঙ্কশনটি নাম যুক্ত অবস্থা ঠিক এই রকম ছিল :

|  |  |
| --- | --- |
| 1  2  3 | def double(x):     return x \* 2  print(double(5)) |

# Module and package

মডিউল হচ্ছে কিছু কোডের সমষ্টি যেখানে বেশ কিছু ফাংশন, ভ্যারিয়েবল বা ডাটা থাকে এবং যেগুলোকে অ্যাক্সেস করে প্রয়োজনে আরেকটি পাইথন প্রোগ্রামে ব্যবহার করা যায়। পাইথনের অনেক অনেক বিল্ট-ইন মডিউল আছে যেগুলোতে অনেক অনেক প্রয়োজনীয় ফাংশন যুক্ত করাই আছে। নিজেদের জন্য কোন প্রোগ্রাম লেখার সময় চাইলে সেই মডিউল গুলো থেকে উক্ত ফিচার গুলো ব্যবহার করা যায়।

নতুন একটি প্রোগ্রামে এরকম কোন মডিউল ব্যবহার করতে চাইলে প্রথমেই সেটিকে import করে নিতে হবে। import MODULE\_NAME এভাবে। এবার এই স্টেটমেন্টের নিচে MODULE\_NAME.VAR এভাবে উক্ত মডিউলের ফাংশন বা ভ্যারিয়েবলকে অ্যাক্সেস করা যাবে। একটি উদাহরণ দেখি -

Example

**import** random,math,datetime

value **=** random.randint(1,5)

print(value)

print(math.sin(30))

print(math.sqrt(16))

print(datetime.datetime.now())

উপরের প্রোগ্রামে value নামের ভ্যারিয়েবলে আমরা একটি র‍্যান্ডম নাম্বার ষ্টোর করতে চেয়েছি। যে র‍্যান্ডম নাম্বারটি হবে ১ থেকে 5 এর মধ্যে। কিন্তু আমরা নিজেরা সেই র‍্যান্ডম নাম্বার তৈরির ফাংশন লিখি নাই। বরং আমরা পাইথনের একটি বিল্ট ইন মডিউল random কে ইম্পোর্ট করে নিয়েছি এবং এর মধ্যে আগেই ডিফাইন করে রাখা randint ফাংশনকে ব্যবহার করে র‍্যান্ডম নাম্বার পাচ্ছি। এ প্রোগ্রামের আউটপুট এক এক বার এক এক রকম আসবে কিন্তু অবশ্যই এমন একটি ভ্যালু প্রিন্ট হবে যার মান ১ থেকে 5 এর মধ্যে।

আরও একভাবে মডিউল ইম্পোর্ট এর কাজ করা যায়। যদি আমাদের কোন একটি মডিউলের নির্দিষ্ট কিছু জিনিষ দরকার হয় তাহলে শুধুমাত্র সেগুলোকে ইম্পোর্ট করা যায়। যেমন নিচের উদাহরণটি -

**from** math **import** sqrt,pi,sin,cos

print(sqrt(16))

print(pi)

print(cos(0))

print(sin(0))

উপরের উদাহরণে আমরা math মডিউল থেকে শুধুমাত্র pi কন্সট্যান্টটি এবং sqrt ফাংশনটিকে ইম্পোর্ট করেছি। আর তাই, এই দুটোকে আমরা ব্যবহার করতে পারছি আমাদের প্রোগ্রামে। এখন ধরুন sqrt নামটা আপনার পছন্দ হচ্ছে না। আপনি চাচ্ছেন square root বের করার ফাংশনের নাম আরেকটু সুন্দর হলে ভালো হয়। সেটাও করতে পারেন নিচের মত করে –

**from** math **import** sqrt **as** root

print(root(16))

কোন একটি মডিউলের সব গুলো অবজেক্ট তথা ফাংশন, ভ্যারিয়েবল, কন্সট্যান্টকে ইম্পোর্ট করার জন্য অনেকেই from MODULE\_NAME import \* ব্যবহার করে থাকেন। এটি একদমই উচিৎ নয়। কারণ এতে করে আপনার কোডের মধ্যে ব্যবহৃত কোন ফাংশন বা ভ্যারিয়েবলের নাম মডিউল থেকে পাওয়া নাকি নিজের তৈরি সেটা নিজেরই বুঝতে সমস্যা হতে পারে।

**from** math **import\***

print(sqrt(16))

print(sin(0))

print(cos(0))

## স্ট্যান্ডার্ড লাইব্রেরী

তিন ধরণের মডিউল হতে পারে। কিছু মডিউল যেগুলো পাইথনের সাথে বিল্ট-ইন আছে, কিছু আছে যেগুলো অন্য কোন ডেভেলপের তৈরি করেছে, এবং কিছু হতে পারে আপনার নিজের তৈরি। প্রথম ধরণের মডিউলকে বলা হয় স্ট্যান্ডার্ড লাইব্রেরি। অনেক অনেক এরকম লাইব্রেরীর মধ্যে কিছু হচ্ছে - string, re, datetime, math, random, os, multiprocessing, subprocess, socket, email, json, doctest, unittest, pdb, argparse এবং sys যেগুলোর মাধ্যমে খুব সহজেই স্ট্রিং পারসিং, ডাটা সিরিয়ালাইজেশন, টেস্টিং, ডিবাগিং, ডেট টাইম নিয়ে কাজ, কমান্ড লাইন আর্গুমেন্ট রিসিভ, ইমেইল পাঠানো ইত্যাদি অনেক অনেক কাজ করা যায়। সত্যি কথা বলতে - পাইথনের এই বিশাল পরিমাণ স্ট্যান্ডার্ড লাইব্রেরির কালেকশনের জন্যও এটি একটি অন্যতম জনপ্রিয় প্রোগ্রামিং ভাষা।

মজার ব্যাপার হচ্ছে কিছু কিছু মডিউল পাইথনে লেখা আবার কিছু কিছু মডিউল সি প্রোগ্রামিং ভাষায় লেখা। [এই লিঙ্কে](https://docs.python.org/3/library/) গেলে পাইথনের স্ট্যান্ডার্ড লাইব্রেরি গুলো সম্পর্কে আরও বিস্তারিত জানা যাবে। আপনাদের সুবিধার্থে লিঙ্ক সহ সেগুলোর লিস্ট নিচেও দেয়া হলঃ

 [1. Introduction](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/intro.html)

 [2. Built-in Functions](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/functions.html)

 [3. Built-in Constants](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/constants.html)

* [3.1. Constants added by the site module](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/constants.html#constants-added-by-the-site-module)

 [4. Built-in Types](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/stdtypes.html)

* [4.1. Truth Value Testing](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/stdtypes.html#truth-value-testing)
* [4.2. Boolean Operations — and, or, not](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/stdtypes.html#boolean-operations-and-or-not)
* [4.3. Comparisons](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/stdtypes.html#comparisons)
* [4.4. Numeric Types — int, float, complex](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/stdtypes.html#numeric-types-int-float-complex)
* [4.5. Iterator Types](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/stdtypes.html#iterator-types)
* [4.6. Sequence Types — list, tuple, range](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/stdtypes.html#sequence-types-list-tuple-range)
* [4.7. Text Sequence Type — str](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/stdtypes.html#text-sequence-type-str)
* [4.8. Binary Sequence Types — bytes, bytearray, memoryview](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/stdtypes.html#binary-sequence-types-bytes-bytearray-memoryview)
* [4.9. Set Types — set, frozenset](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/stdtypes.html#set-types-set-frozenset)
* [4.10. Mapping Types — dict](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/stdtypes.html#mapping-types-dict)
* [4.11. Context Manager Types](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/stdtypes.html#context-manager-types)
* [4.12. Other Built-in Types](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/stdtypes.html#other-built-in-types)
* [4.13. Special Attributes](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/stdtypes.html#special-attributes)

 [5. Built-in Exceptions](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/exceptions.html)

* [5.1. Base classes](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/exceptions.html#base-classes)
* [5.2. Concrete exceptions](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/exceptions.html#concrete-exceptions)
* [5.3. Warnings](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/exceptions.html#warnings)
* [5.4. Exception hierarchy](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/exceptions.html#exception-hierarchy)

 [6. Text Processing Services](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/text.html)

* [6.1. string — Common string operations](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/string.html)
* [6.2. re — Regular expression operations](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/re.html)
* [6.3. difflib — Helpers for computing deltas](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/difflib.html)
* [6.4. textwrap — Text wrapping and filling](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/textwrap.html)
* [6.5. unicodedata — Unicode Database](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/unicodedata.html)
* [6.6. stringprep — Internet String Preparation](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/stringprep.html)
* [6.7. readline — GNU readline interface](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/readline.html)
* [6.8. rlcompleter — Completion function for GNU readline](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/rlcompleter.html)

 [7. Binary Data Services](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/binary.html)

* [7.1. struct — Interpret bytes as packed binary data](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/struct.html)
* [7.2. codecs — Codec registry and base classes](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/codecs.html)

 [8. Data Types](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/datatypes.html)

* [8.1. datetime — Basic date and time types](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/datetime.html)
* [8.2. calendar — General calendar-related functions](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/calendar.html)
* [8.3. collections — Container datatypes](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/collections.html)
* [8.4. collections.abc — Abstract Base Classes for Containers](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/collections.abc.html)
* [8.5. heapq — Heap queue algorithm](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/heapq.html)
* [8.6. bisect — Array bisection algorithm](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/bisect.html)
* [8.7. array — Efficient arrays of numeric values](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/array.html)
* [8.8. weakref — Weak references](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/weakref.html)
* [8.9. types — Dynamic type creation and names for built-in types](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/types.html)
* [8.10. copy — Shallow and deep copy operations](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/copy.html)
* [8.11. pprint — Data pretty printer](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/pprint.html)
* [8.12. reprlib — Alternate repr() implementation](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/reprlib.html)
* [8.13. enum — Support for enumerations](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/enum.html)

 [9. Numeric and Mathematical Modules](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/numeric.html)

* [9.1. numbers — Numeric abstract base classes](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/numbers.html)
* [9.2. math — Mathematical functions](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/math.html)
* [9.3. cmath — Mathematical functions for complex numbers](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/cmath.html)
* [9.4. decimal — Decimal fixed point and floating point arithmetic](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/decimal.html)
* [9.5. fractions — Rational numbers](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/fractions.html)
* [9.6. random — Generate pseudo-random numbers](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/random.html)
* [9.7. statistics — Mathematical statistics functions](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/statistics.html)

 [10. Functional Programming Modules](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/functional.html)

* [10.1. itertools — Functions creating iterators for efficient looping](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/itertools.html)
* [10.2. functools — Higher-order functions and operations on callable objects](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/functools.html)
* [10.3. operator — Standard operators as functions](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/operator.html)

 [11. File and Directory Access](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/filesys.html)

* [11.1. pathlib — Object-oriented filesystem paths](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/pathlib.html)
* [11.2. os.path — Common pathname manipulations](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/os.path.html)
* [11.3. fileinput — Iterate over lines from multiple input streams](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/fileinput.html)
* [11.4. stat — Interpreting stat() results](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/stat.html)
* [11.5. filecmp — File and Directory Comparisons](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/filecmp.html)
* [11.6. tempfile — Generate temporary files and directories](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/tempfile.html)
* [11.7. glob — Unix style pathname pattern expansion](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/glob.html)
* [11.8. fnmatch — Unix filename pattern matching](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/fnmatch.html)
* [11.9. linecache — Random access to text lines](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/linecache.html)
* [11.10. shutil — High-level file operations](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/shutil.html)
* [11.11. macpath — Mac OS 9 path manipulation functions](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/macpath.html)

 [12. Data Persistence](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/persistence.html)

* [12.1. pickle — Python object serialization](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/pickle.html)
* [12.2. copyreg — Register pickle support functions](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/copyreg.html)
* [12.3. shelve — Python object persistence](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/shelve.html)
* [12.4. marshal — Internal Python object serialization](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/marshal.html)
* [12.5. dbm — Interfaces to Unix “databases”](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/dbm.html)
* [12.6. sqlite3 — DB-API 2.0 interface for SQLite databases](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/sqlite3.html)

 [13. Data Compression and Archiving](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/archiving.html)

* [13.1. zlib — Compression compatible with **gzip**](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/zlib.html)
* [13.2. gzip — Support for **gzip** files](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/gzip.html)
* [13.3. bz2 — Support for **bzip2** compression](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/bz2.html)
* [13.4. lzma — Compression using the LZMA algorithm](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/lzma.html)
* [13.5. zipfile — Work with ZIP archives](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/zipfile.html)
* [13.6. tarfile — Read and write tar archive files](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/tarfile.html)

 [14. File Formats](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/fileformats.html)

* [14.1. csv — CSV File Reading and Writing](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/csv.html)
* [14.2. configparser — Configuration file parser](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/configparser.html)
* [14.3. netrc — netrc file processing](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/netrc.html)
* [14.4. xdrlib — Encode and decode XDR data](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/xdrlib.html)
* [14.5. plistlib — Generate and parse Mac OS X .plist files](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/plistlib.html)

 [15. Cryptographic Services](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/crypto.html)

* [15.1. hashlib — Secure hashes and message digests](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/hashlib.html)
* [15.2. hmac — Keyed-Hashing for Message Authentication](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/hmac.html)

 [16. Generic Operating System Services](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/allos.html)

* [16.1. os — Miscellaneous operating system interfaces](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/os.html)
* [16.2. io — Core tools for working with streams](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/io.html)
* [16.3. time — Time access and conversions](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/time.html)
* [16.4. argparse — Parser for command-line options, arguments and sub-commands](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/argparse.html)
* [16.5. getopt — C-style parser for command line options](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/getopt.html)
* [16.6. logging — Logging facility for Python](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/logging.html)
* [16.7. logging.config — Logging configuration](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/logging.config.html)
* [16.8. logging.handlers — Logging handlers](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/logging.handlers.html)
* [16.9. getpass — Portable password input](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/getpass.html)
* [16.10. curses — Terminal handling for character-cell displays](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/curses.html)
* [16.11. curses.textpad — Text input widget for curses programs](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/curses.html#module-curses.textpad)
* [16.12. curses.ascii — Utilities for ASCII characters](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/curses.ascii.html)
* [16.13. curses.panel — A panel stack extension for curses](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/curses.panel.html)
* [16.14. platform — Access to underlying platform’s identifying data](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/platform.html)
* [16.15. errno — Standard errno system symbols](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/errno.html)
* [16.16. ctypes — A foreign function library for Python](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/ctypes.html)

 [17. Concurrent Execution](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/concurrency.html)

* [17.1. threading — Thread-based parallelism](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/threading.html)
* [17.2. multiprocessing — Process-based parallelism](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/multiprocessing.html)
* [17.3. The concurrent package](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/concurrent.html)
* [17.4. concurrent.futures — Launching parallel tasks](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/concurrent.futures.html)
* [17.5. subprocess — Subprocess management](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/subprocess.html)
* [17.6. sched — Event scheduler](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/sched.html)
* [17.7. queue — A synchronized queue class](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/queue.html)
* [17.8. dummy\_threading — Drop-in replacement for the threading module](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/dummy_threading.html)
* [17.9. \_thread — Low-level threading API](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/_thread.html)
* [17.10. \_dummy\_thread — Drop-in replacement for the \_thread module](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/_dummy_thread.html)

 [18. Interprocess Communication and Networking](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/ipc.html)

* [18.1. socket — Low-level networking interface](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/socket.html)
* [18.2. ssl — TLS/SSL wrapper for socket objects](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/ssl.html)
* [18.3. select — Waiting for I/O completion](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/select.html)
* [18.4. selectors – High-level I/O multiplexing](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/selectors.html)
* [18.5. asyncio – Asynchronous I/O, event loop, coroutines and tasks](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/asyncio.html)
* [18.6. asyncore — Asynchronous socket handler](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/asyncore.html)
* [18.7. asynchat — Asynchronous socket command/response handler](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/asynchat.html)
* [18.8. signal — Set handlers for asynchronous events](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/signal.html)
* [18.9. mmap — Memory-mapped file support](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/mmap.html)

 [19. Internet Data Handling](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/netdata.html)

* [19.1. email — An email and MIME handling package](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/email.html)
* [19.2. json — JSON encoder and decoder](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/json.html)
* [19.3. mailcap — Mailcap file handling](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/mailcap.html)
* [19.4. mailbox — Manipulate mailboxes in various formats](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/mailbox.html)
* [19.5. mimetypes — Map filenames to MIME types](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/mimetypes.html)
* [19.6. base64 — Base16, Base32, Base64, Base85 Data Encodings](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/base64.html)
* [19.7. binhex — Encode and decode binhex4 files](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/binhex.html)
* [19.8. binascii — Convert between binary and ASCII](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/binascii.html)
* [19.9. quopri — Encode and decode MIME quoted-printable data](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/quopri.html)
* [19.10. uu — Encode and decode uuencode files](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/uu.html)

 [20. Structured Markup Processing Tools](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/markup.html)

* [20.1. html — HyperText Markup Language support](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/html.html)
* [20.2. html.parser — Simple HTML and XHTML parser](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/html.parser.html)
* [20.3. html.entities — Definitions of HTML general entities](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/html.entities.html)
* [20.4. XML Processing Modules](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/xml.html)
* [20.5. xml.etree.ElementTree — The ElementTree XML API](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/xml.etree.elementtree.html)
* [20.6. xml.dom — The Document Object Model API](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/xml.dom.html)
* [20.7. xml.dom.minidom — Minimal DOM implementation](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/xml.dom.minidom.html)
* [20.8. xml.dom.pulldom — Support for building partial DOM trees](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/xml.dom.pulldom.html)
* [20.9. xml.sax — Support for SAX2 parsers](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/xml.sax.html)
* [20.10. xml.sax.handler — Base classes for SAX handlers](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/xml.sax.handler.html)
* [20.11. xml.sax.saxutils — SAX Utilities](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/xml.sax.utils.html)
* [20.12. xml.sax.xmlreader — Interface for XML parsers](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/xml.sax.reader.html)
* [20.13. xml.parsers.expat — Fast XML parsing using Expat](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/pyexpat.html)

 [21. Internet Protocols and Support](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/internet.html)

* [21.1. webbrowser — Convenient Web-browser controller](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/webbrowser.html)
* [21.2. cgi — Common Gateway Interface support](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/cgi.html)
* [21.3. cgitb — Traceback manager for CGI scripts](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/cgitb.html)
* [21.4. wsgiref — WSGI Utilities and Reference Implementation](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/wsgiref.html)
* [21.5. urllib — URL handling modules](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/urllib.html)
* [21.6. urllib.request — Extensible library for opening URLs](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/urllib.request.html)
* [21.7. urllib.response — Response classes used by urllib](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/urllib.request.html#module-urllib.response)
* [21.8. urllib.parse — Parse URLs into components](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/urllib.parse.html)
* [21.9. urllib.error — Exception classes raised by urllib.request](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/urllib.error.html)
* [21.10. urllib.robotparser — Parser for robots.txt](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/urllib.robotparser.html)
* [21.11. http — HTTP modules](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/http.html)
* [21.12. http.client — HTTP protocol client](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/http.client.html)
* [21.13. ftplib — FTP protocol client](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/ftplib.html)
* [21.14. poplib — POP3 protocol client](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/poplib.html)
* [21.15. imaplib — IMAP4 protocol client](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/imaplib.html)
* [21.16. nntplib — NNTP protocol client](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/nntplib.html)
* [21.17. smtplib — SMTP protocol client](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/smtplib.html)
* [21.18. smtpd — SMTP Server](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/smtpd.html)
* [21.19. telnetlib — Telnet client](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/telnetlib.html)
* [21.20. uuid — UUID objects according to RFC 4122](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/uuid.html)
* [21.21. socketserver — A framework for network servers](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/socketserver.html)
* [21.22. http.server — HTTP servers](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/http.server.html)
* [21.23. http.cookies — HTTP state management](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/http.cookies.html)
* [21.24. http.cookiejar — Cookie handling for HTTP clients](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/http.cookiejar.html)
* [21.25. xmlrpc — XMLRPC server and client modules](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/xmlrpc.html)
* [21.26. xmlrpc.client — XML-RPC client access](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/xmlrpc.client.html)
* [21.27. xmlrpc.server — Basic XML-RPC servers](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/xmlrpc.server.html)
* [21.28. ipaddress — IPv4/IPv6 manipulation library](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/ipaddress.html)

 [22. Multimedia Services](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/mm.html)

* [22.1. audioop — Manipulate raw audio data](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/audioop.html)
* [22.2. aifc — Read and write AIFF and AIFC files](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/aifc.html)
* [22.3. sunau — Read and write Sun AU files](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/sunau.html)
* [22.4. wave — Read and write WAV files](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/wave.html)
* [22.5. chunk — Read IFF chunked data](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/chunk.html)
* [22.6. colorsys — Conversions between color systems](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/colorsys.html)
* [22.7. imghdr — Determine the type of an image](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/imghdr.html)
* [22.8. sndhdr — Determine type of sound file](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/sndhdr.html)
* [22.9. ossaudiodev — Access to OSS-compatible audio devices](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/ossaudiodev.html)

 [23. Internationalization](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/i18n.html)

* [23.1. gettext — Multilingual internationalization services](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/gettext.html)
* [23.2. locale — Internationalization services](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/locale.html)

 [24. Program Frameworks](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/frameworks.html)

* [24.1. turtle — Turtle graphics](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/turtle.html)
* [24.2. cmd — Support for line-oriented command interpreters](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/cmd.html)
* [24.3. shlex — Simple lexical analysis](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/shlex.html)

 [25. Graphical User Interfaces with Tk](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/tk.html)

* [25.1. tkinter — Python interface to Tcl/Tk](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/tkinter.html)
* [25.2. tkinter.ttk — Tk themed widgets](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/tkinter.ttk.html)
* [25.3. tkinter.tix — Extension widgets for Tk](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/tkinter.tix.html)
* [25.4. tkinter.scrolledtext — Scrolled Text Widget](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/tkinter.scrolledtext.html)
* [25.5. IDLE](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/idle.html)
* [25.6. Other Graphical User Interface Packages](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/othergui.html)

 [26. Development Tools](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/development.html)

* [26.1. typing — Support for type hints](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/typing.html)
* [26.2. pydoc — Documentation generator and online help system](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/pydoc.html)
* [26.3. doctest — Test interactive Python examples](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/doctest.html)
* [26.4. unittest — Unit testing framework](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/unittest.html)
* [26.5. unittest.mock — mock object library](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/unittest.mock.html)
* [26.6. unittest.mock — getting started](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/unittest.mock-examples.html)
* [26.7. 2to3 - Automated Python 2 to 3 code translation](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/2to3.html)
* [26.8. test — Regression tests package for Python](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/test.html)
* [26.9. test.support — Utilities for the Python test suite](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/test.html#module-test.support)

 [27. Debugging and Profiling](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/debug.html)

* [27.1. bdb — Debugger framework](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/bdb.html)
* [27.2. faulthandler — Dump the Python traceback](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/faulthandler.html)
* [27.3. pdb — The Python Debugger](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/pdb.html)
* [27.4. The Python Profilers](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/profile.html)
* [27.5. timeit — Measure execution time of small code snippets](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/timeit.html)
* [27.6. trace — Trace or track Python statement execution](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/trace.html)
* [27.7. tracemalloc — Trace memory allocations](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/tracemalloc.html)

 [28. Software Packaging and Distribution](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/distribution.html)

* [28.1. distutils — Building and installing Python modules](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/distutils.html)
* [28.2. ensurepip — Bootstrapping the pip installer](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/ensurepip.html)
* [28.3. venv — Creation of virtual environments](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/venv.html)
* [28.4. zipapp — Manage executable python zip archives](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/zipapp.html)

 [29. Python Runtime Services](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/python.html)

* [29.1. sys — System-specific parameters and functions](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/sys.html)
* [29.2. sysconfig — Provide access to Python’s configuration information](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/sysconfig.html)
* [29.3. builtins — Built-in objects](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/builtins.html)
* [29.4. \_\_main\_\_ — Top-level script environment](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/__main__.html)
* [29.5. warnings — Warning control](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/warnings.html)
* [29.6. contextlib — Utilities for with-statement contexts](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/contextlib.html)
* [29.7. abc — Abstract Base Classes](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/abc.html)
* [29.8. atexit — Exit handlers](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/atexit.html)
* [29.9. traceback — Print or retrieve a stack traceback](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/traceback.html)
* [29.10. \_\_future\_\_ — Future statement definitions](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/__future__.html)
* [29.11. gc — Garbage Collector interface](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/gc.html)
* [29.12. inspect — Inspect live objects](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/inspect.html)
* [29.13. site — Site-specific configuration hook](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/site.html)
* [29.14. fpectl — Floating point exception control](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/fpectl.html)

 [30. Custom Python Interpreters](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/custominterp.html)

* [30.1. code — Interpreter base classes](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/code.html)
* [30.2. codeop — Compile Python code](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/codeop.html)

 [31. Importing Modules](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/modules.html)

* [31.1. zipimport — Import modules from Zip archives](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/zipimport.html)
* [31.2. pkgutil — Package extension utility](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/pkgutil.html)
* [31.3. modulefinder — Find modules used by a script](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/modulefinder.html)
* [31.4. runpy — Locating and executing Python modules](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/runpy.html)
* [31.5. importlib – The implementation of import](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/importlib.html)

 [32. Python Language Services](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/language.html)

* [32.1. parser — Access Python parse trees](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/parser.html)
* [32.2. ast — Abstract Syntax Trees](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/ast.html)
* [32.3. symtable — Access to the compiler’s symbol tables](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/symtable.html)
* [32.4. symbol — Constants used with Python parse trees](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/symbol.html)
* [32.5. token — Constants used with Python parse trees](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/token.html)
* [32.6. keyword — Testing for Python keywords](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/keyword.html)
* [32.7. tokenize — Tokenizer for Python source](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/tokenize.html)
* [32.8. tabnanny — Detection of ambiguous indentation](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/tabnanny.html)
* [32.9. pyclbr — Python class browser support](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/pyclbr.html)
* [32.10. py\_compile — Compile Python source files](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/py_compile.html)
* [32.11. compileall — Byte-compile Python libraries](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/compileall.html)
* [32.12. dis — Disassembler for Python bytecode](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/dis.html)
* [32.13. pickletools — Tools for pickle developers](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/pickletools.html)

 [33. Miscellaneous Services](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/misc.html)

* [33.1. formatter — Generic output formatting](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/formatter.html)

 [34. MS Windows Specific Services](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/windows.html)

* [34.1. msilib — Read and write Microsoft Installer files](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/msilib.html)
* [34.2. msvcrt – Useful routines from the MS VC++ runtime](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/msvcrt.html)
* [34.3. winreg – Windows registry access](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/winreg.html)
* [34.4. winsound — Sound-playing interface for Windows](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/winsound.html)

 [35. Unix Specific Services](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/unix.html)

* [35.1. posix — The most common POSIX system calls](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/posix.html)
* [35.2. pwd — The password database](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/pwd.html)
* [35.3. spwd — The shadow password database](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/spwd.html)
* [35.4. grp — The group database](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/grp.html)
* [35.5. crypt — Function to check Unix passwords](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/crypt.html)
* [35.6. termios — POSIX style tty control](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/termios.html)
* [35.7. tty — Terminal control functions](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/tty.html)
* [35.8. pty — Pseudo-terminal utilities](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/pty.html)
* [35.9. fcntl — The fcntl and ioctl system calls](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/fcntl.html)
* [35.10. pipes — Interface to shell pipelines](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/pipes.html)
* [35.11. resource — Resource usage information](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/resource.html)
* [35.12. nis — Interface to Sun’s NIS (Yellow Pages)](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/nis.html)
* [35.13. syslog — Unix syslog library routines](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/syslog.html)

 [36. Superseded Modules](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/superseded.html)

* [36.1. optparse — Parser for command line options](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/optparse.html)
* [36.2. imp — Access the import internals](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/imp.html)

 [37. Undocumented Modules](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/undoc.html)

* [37.1. Platform specific modules](https://github.com/howtocode-com-bd/python.howtocode.com.bd/tree/984867888d1013767cc9b1dfd5527bf60445a67c/undoc.html#platform-specific-modules)

## pip

আগের চ্যাপ্টারেই আমরা বলেছি পাইথনের বিল্ট ইন মডিউলের সাথে সাথে অন্যদের ডেভেলপ করা অনেক মডিউলও আছে যেগুলো পাইথন প্রোগ্রামিং -কে করেছে আরও প্রোডাক্টিভ এবং সহজ। সেরকম অন্যদের ডেভেলপ করা মডিউল গুলোকে পাওয়া যায় [PyPI - the Python Package Index](https://pypi.python.org/pypi) এখানে।

এখানে জমা থাকা মডিউল গুলোকে নিজের কম্পিউটারে ইন্সটল করার সবচেয়ে সহজ পদ্ধতি হচ্ছে **pip** নামের একটি টুল বা প্রোগ্রাম ব্যবহার করা। যদি আপনি পাইথনের অফিসিয়াল সাইট থেকে পাইথনের আপডেটেড ভার্সন ডাউনলোড করে ইন্সটল করে থাকেন তাহলে এই টুলটিও সাথে ইন্সটল হয়ে থাকার কথা।

তো যাই হোক, এই টুল ব্যবহার করে উপরোক্ত পাইথন প্যাকেজ ইনডেক্স সাইট থেকে কোন লাইব্রেরী বা মডিউলকে ইন্সটল করার সহজ পদ্ধতি হচ্ছে - প্রথমে টার্মিনাল ওপেন করতে হবে (উইন্ডোজ হলে কমান্ড প্রম্পট) এবং নিচের কমান্ডটি ইস্যু করতে হবে,

pip install LIBRARY\_NAME

example: pip install bs4

লাইব্রেরীর নাম দেখে নিতে হবে ওই সাইট থেকেই। যেমন মেশিন লার্নিং এবং ডাটা মাইনিং এর জন্য বহুল ব্যবহৃত [মডিউল সেট](https://pypi.python.org/pypi/scikit-learn/0.18) scikit-learn কে ইন্সটল করা যাবে নিচের মত করে,

এভাবে ইন্সটল করার পর ওই লাইব্রেরী বা মডিউলকে নিজের প্রোগ্রামে import করে নিতে হবে।

## প্যাকেজিং

পাইথনে প্যাকেজিং বলতে বুঝায়, আপনার লেখা বিভিন্ন মডিউল গুলোকে একটা স্ট্যান্ডার্ড ফরম্যাটে একত্র করে বান্ডেল করা যাতে অন্য ইউজাররা খুব সহজেই আপনার বানানো প্রোগ্রামকে ব্যবহার করতে পারে। যেমন ধরুন, ওয়েব থেকে ডাটা স্ক্র্যাপ করার জন্য যাবতীয় সব রকম ফাংশনালিটি গুলো বানিয়ে কিছু ডেভেলপার একটি প্যাকেজ রিলিজ দিয়েছে যার নাম BeautifulSoup. আর আমরা খুব সহজেই সেই প্যাকজেটি আমাদের প্রজেক্টে ইন্সটল করে তার বিভিন্ন রেডিমেড ফাংশনকে ব্যবহার করতে পারি।

নিচে আমরা ধাপে ধাপে দেখবো কিভাবে একটি কাস্টম প্যাকেজকে [PyPI](https://pypi.python.org/pypi) তথা Python Package Index -এ আপলোড করতে হয়। ধরে নিচ্ছি, আমাদের প্যাকেজের সোর্স কোড [github](https://github.com/) এ হোস্ট করা আছে এবং প্যাকেজটির নাম mypackage.

প্রথমেই, [PyPI Live](http://pypi.python.org/pypi?%3Aaction=register_form) এবং [PyPI Test](http://testpypi.python.org/pypi?%3Aaction=register_form) -এ আপনার একাউন্ট খুলে নিতে পারেন।

.pypirc **কনফিগারেশন ফাইল তৈরি** এই ফাইলে PyPI এর সাথে আপনার অথেন্টিকেশনের কনফিগারেশন গুলো উল্লেখ থাকবে।

[distutils]

index**-**servers **=**

    pypi

    pypitest

[pypi]

repository**=**https:**//**pypi.python.org**/**pypi

username**=**your\_username

password**=**your\_password

[pypitest]

repository**=**https:**//**testpypi.python.org**/**pypi

username**=**your\_username

password**=**your\_password

কাজের সুবিধার জন্য এই ফাইলটিকে আপনার হোম ফোল্ডারে রাখতে পারেন। অর্থাৎ যার লোকেশন হবে ~/.pypirc

যেহেতু এই ফাইলে আপনার কিছু সংবেদনশীল তথ্য যেমন পাসওয়ার্ড থাকবে তাই এই ফাইলের পারমিশন বদলে নিতে পারেন - chmod 600 ~/.pypirc এই কমান্ড ইস্যু করে।

**প্যাকেজ কন্টেন্ট তৈরি**

ইউজারের জন্য এভাবে প্যাকেজ তৈরি করতে গিয়ে setuptools এবং distutils মডিউলের দরকার পরে। প্রথমেই একটি ডিরেক্টরির মধ্যে সব গুলো প্রোগ্রাম ফাইলকে রাখতে হবে। ওই একই ডিরেক্টরির মধ্যে \_\_init\_\_.py নামের একটি ফাইলও রাখতে হবে। এটা নিয়ম। এর উপস্থিতির মাধ্যমে পাইথন এটাকে একটা প্যাকেজ হিসেবে ধরে নেয়। এই ফাইলটি ফাকাও হতে পারে।

এরপর, আপনার প্রোগ্রাম ফাইলগুলো এবং এই স্পেশাল ইনিসিয়ালাইজার ফাইল সহ ডিরেক্টরিটিকে আরও একটি রুট ডিরেক্টরির মধ্যে রাখতে হবে। এবং সেই রুট ডিরেক্টরির মধ্যে একটি readme ফাইল, একটি লাইসেন্স ফাইল এবং গুরুত্বপূর্ণ একটি ফাইল setup.py কে রাখতে হবে।

ডিরেক্টরি লিস্টিং উদাহরণ,

MyPackage/ # Name of this dir can be anything

LICENSE.txt

README.md

setup.py

setup.cfg

mypackage/ # actual package name

\_\_init\_\_.py

myfile.py

anotherfile.py

setup.py ফাইলে কিছু গুরুত্বপূর্ণ তথ্য যুক্ত করতে হয় যাতে করে আপনার ডেভেলপ করা প্যাকেজটি [PyPI](https://pypi.python.org/pypi) তথা Python Package Index এ আপলোড করার উপযোগী হয়। এতে করে ইউজাররা [pip](https://pypi.python.org/pypi/pip) টুল ইউজ করে খুব সহজেই আপনার প্যাকেজটি ইন্সটল করে নিতে পারবে।

ফাইলটি হতে পারে নিচের মত,

from distutils.core import setup

setup(

name = 'mypackage',

packages = ['mypackage'], # this must be the same as the name above

version = '0.1',

description = 'A random test lib',

author = 'Nuhil Mehdy',

author\_email = 'nuhil@nuhil.net',

url = 'https://github.com/nuhil/mypackage', # use the URL to the github repo

download\_url = 'https://github.com/nuhil/mypackage/archive/0.1.tar.gz', # I'll explain this in a second

keywords = ['testing', 'logging', 'example'], # arbitrary keywords

classifiers = [],

)

download\_url লিঙ্কের মাধ্যমে আপনার প্যাকেজের টারবল ভার্সন পয়েন্ট করে দিতে হয়। যদি আপনার সোর্স কোড [github](https://github.com/) -এ হোস্ট করা থাকে এবং মুল রিপোজিটরি থেকে একটি ট্যাগ তৈরি করা থাকে তাহলে সেখানে এরকম টারবল রেডি থাকে।

**ট্যাগ তৈরি করতেঃ** প্রথমে লোকাল রিপোজিটরিতে ঢুকে কমান্ড দিতে হবে git tag 0.1 -m "Adds a tag so that we can put this on PyPI." এরপর git tag কমান্ড দিয়ে ট্যাগ গুলো দেখা যাবে। আগের কমান্ড দিয়ে ঠিক ঠাক ট্যাগ তৈরি হয়ে থাকলে লিস্ট আসবে এমন - 0.1. এরপর git push --tags origin master কমান্ড দিতে হবে যার মাধ্যমে github এ এই ট্যাগের তথ্য যুক্ত হয়ে যাবে। আর সেই নির্দিষ্ট ট্যাগ ওয়ালা টারবলটী ডাউনলোডের জন্য তৈরি থাকবে এই লিঙ্কে - https://github.com/{username}/{module\_name}/archive/{tag}.tar.gz

**setup.cfg ফাইল** যদি আপনার readme ফাইলটি মার্কডাউন ফরম্যাটে লেখা হয়ে থাকে তাহলে এই ফাইলটি দরকার পরে। এর কন্টেন্ট হবে নিচের মত।

[metadata]

description-file = README.md

.txt ফরম্যাটে লেখা readme ফাইল হলে উপরোক্ত ফাইল যুক্ত করার দরকার পরে না।

**PyPI Test এ প্যাকেজ আপলোড** প্রথমে নিচের কমান্ড ইস্যু করুন -

python setup.py register -r pypitest

এরপর,

python setup.py sdist upload -r pypitest

সব কিছু ঠিক ঠাক থাকলে আপনার প্যাকেজকে [Test PyPI Repository](https://testpypi.python.org/pypi) তে দেখা যাবে।

**PyPI Live এ প্যাকেজ আপলোড** প্রথমে নিচের কমান্ড ইস্যু করুন -

python setup.py register -r pypi

এরপর,

python setup.py sdist upload -r pypi

## মডিউল

0 1 1 2 3 5 8 13 21 34 55 89 144 233 – এই ধারার সংখ্যাগুলোর মাঝে একটা চমৎকার ব্যাপার আছে। ব্যাপারটা হলো এই ধারার প্রতিটি সংখ্যা এর আগের দুটি সংখ্যার যোগফল। ইতালিয়ান গণিতবিদ লিওনার্দো ফিবোনাচি এটি প্রকাশ করেন। তার নামানুসারে এই ধারাকে বলা হয় ফিবোনাচি ধারা। সাধারণত ০ ও ১ অথবা ১ ও ১ দিয়ে ফিবোনাচি ধারা শুরু হয়।

যাহোক, আমরা এখন কোনো একটা নির্দিষ্ট ডিরেক্টরিতে বা ফোল্ডারে fibo.py নামে একটা ফাইল তৈরি করব। এবার সেখানে ফিবোনাচি ধারা বের করার কোড লিখব।

def fib(n):

series = []

a, b = 0, 1

while b < n:

series.append(b)

a, b = b, a+b

return series

if \_\_name\_\_ == "\_\_main\_\_":

temp = fib(100)

print(temp)

**আউটপুট**

[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

এবার আমরা আমাদের পুরনো সেই test.py তে ফিরে যাই। সেখানে আমরা একটা ফিবোনাচি সিরিজের লিস্টকে for লুপ দিয়ে ইটারেট করে প্রিন্ট করব।

import fibo

series = fibo.fib(100)

for item in series:

print(item)

**আউটপুট**

1

1

2

3

5

8

13

21

34

55

89

ম্যাজিকটা খেয়াল করলাম সবাই? fibo.py তে লেখা fib() ফাংশনটাকেই আবার ব্যবহার করেছি আমরা। এই যে ফাংশনটাকে আবার না লিখে বা কপি না করেই ব্যবহার করলাম এটাই হল মডিউলের সবচেয়ে বড় সুবিধা। মডিউল হল লজিকালি কোড অর্গানাইজ করার উপায়। পাইথনিক ভাষায়, মডিউল হল একটা ফাইল যেখানে পাইথন ডেফিনিশন ও স্টেটমেন্ট থাকে। .py অংশ বাদ দিলে এই ফাইলই হচ্ছে মডিউলের নাম। import স্টেটমেন্টের মাধ্যমে আমরা একটা মডিউলকে সোর্স ফাইলে আমদানি করতে পারি, সোজা কথায় যোগ করতে পারি। আর মডিউলের নামের পর ডট . চিহ্ন দিয়ে আমরা এর বিভিন্ন অ্যাট্রিবিউটকে ব্যবহার বা অ্যাক্সেস করতে পারি।

আমরা চাইলে সরাসরি কোন মডিউল থেকে কোন নির্দিষ্ট অ্যাট্রিবিউটকে আমদানি করতে পারি। এজন্য আমাদের from ... import ... স্টেটমেন্ট ব্যবহার করতে হবে।

from fibo import fib

series = fib(100)

for item in series:

print(item)

**আউটপুট**

1

1

2

3

5

8

13

21

34

55

89

একটা মডিউলের সবগুলো অ্যাট্রিবিউটকে একবারে আমদানি করার জন্য from ... import \* স্টেটমেন্ট ব্যবহার করতে পারি আমরা।

from fibo import \*

series = fib(100)

for item in series:

print(item)

**আউটপুট**

1

1

2

3

5

8

13

21

34

55

89

এবার আসি if \_\_name\_\_ == "\_\_main\_\_" এর কাহিনিতে। যখন আমরা সরাসরি একটা স্ক্রিপ্ট রান করি কেবল ও কেবলমাত্র তখনই এর \_\_name\_\_, ‍"\_\_main\_\_" এর সমান হয়। আর এই ব্লকের কোড রান হবার সুযোগ পায়। পাইথনে প্রতিটি স্ক্রিপ্টই এক হিসাবে মডিউল। যখন একটা স্ক্রিপ্টে লেখা ক্লাস বা ফাংশনকে আমরা অন্য কোথাও আমদানি করে কাজ করতে যাব তখন ব্যাকগ্রাউন্ডে পুরো স্ক্রিপ্টটাই রান করবে ও আউটপুট তৈরি করবে। কিন্তু তা তো আমাদের দরকার নাই। এইজন্যই if \_\_name\_\_ == "\_\_main\_\_" ব্যবহার করা হয়। মডিউল আমদানি করার ক্ষেত্রে ঐ মডিউলের if \_\_name\_\_ == "\_\_main\_\_" এর অধীনে থাকা কোড রান হবে না। কারণ তখন \_\_name\_\_ আর "\_\_main\_\_" সমান হয় না। এজন্য পারফেক্ট ভাবে পাইথন স্ক্রিপ্ট লেখার উপায় হল:

def my\_function():

# do something

def your\_function():

# do something

def main():

# call all functions here

my\_function()

# play with them

your\_function()

if \_\_name\_\_ == "\_\_main\_\_":

# now call main function

main()

### মডিউল সার্চ পাথ

যখন আমরা fibo মডিউলটাকে আমদানি করেছি তখন পাইথন ইন্টারপ্রিটার fibo নামের একটা বিল্ট-ইন ফাংশনের খোঁজ করে প্রথমে। না পেলে, বিভিন্ন ডিরেক্টরিতে fibo.py নামের একটা ফাইলের খোঁজ করে। এই ডিরেক্টরিগুলোর লিস্ট sys.path ভ্যারিয়েবলে থাকে। এই ডিরেক্টরি সার্চিংয়ের ব্যাপারটাও নিয়ম মেনে হয়। রানিং স্ক্রিপ্টটা যে ডিরেক্টরিতে আছে ইন্টারপ্রিটার প্রথমে সেখানে খোঁজ করে। তারপর শেল ভ্যারিয়েবল PYTHONPATH এর ডিরেক্টরি লিস্টে সার্চ করে। আর সবার শেষে ইন্সটলেশন-ডিপেন্ডেন্ট ডিফল্ট পাথে সার্চ করে। তারপরও খুঁজে না পেলে ImportError এক্সেপশান থ্রো করে।

from omuk import tomuk

### স্টান্ডার্ড লাইব্রেরি

ডিফল্টভাবেই পাইথনের সাথে অনেকগুলো মডিউল থাকে। এদেরকে স্টান্ডার্ড লাইব্রেরি বলে। কেউ কেউ অবশ্য স্টান্ডার্ড মডিউলও বলে। যেমন: time একটা স্টান্ডার্ড মডিউল। বিল্ট-ইন ফাংশন dir() দিয়ে আমরা এই মডিউলের সবগুলো ভ্যালিড অ্যাট্রিবিউট দেখতে পারি।

**import** time

print(dir(time))

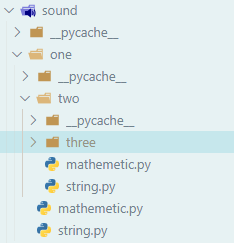
### প্যাকেজ

প্যাকেজ হল মডিউল স্ট্রাকচার করার একটা উপায়। মডিউল, সাব-প্যাকেজ, সাব-সাব-প্যাকেজ ইত্যাদি নিয়ে একটা প্যাকেজ হয়। পাইথনের অফিসিয়াল ডক থেকে আমরা একটা উদাহরণ দেখব।



এখানে আমরা sound প্যাকেজের বিভিন্ন সাব-প্যাকেজ ও মডিউল দেখতে পাচ্ছি। পাইথন যাতে ভিতরের ডিরেক্টরিগুলোকে সাব-প্যাকেজ মনে করে সেজন্য \_\_init\_\_.py ফাইল থাকা দরকার। \_\_init\_\_.py ফাইলে ইনিশিয়ালাইজেশন কোড থাকতে পারে। আবার খালি থাকলেও কোন সমস্যা নাই।

উপরের প্যাকেজ থেকে আমরা এখন echo মডিউলটা আমদানি করব। আর echo সাব-প্যাকেজের ফাংশন echofilter() ব্যবহার করব।



**from** sound.one.two.three.mathemetic **import** fun1

fun1(2,2)

একটা ব্যাপার জানা থাকা উচিত আমাদের। from package import item ব্যবহার করার ক্ষেত্রে item সাব-মডিউলও হতে পারে আবার সাব-প্যাকেজও হতে পারে অথবা প্যাকেজে ডিফাইন করা কোন ফাংশন, ক্লাস বা ভ্যারিয়েবল। আর import item.subitem.subsubitem ব্যবহার করার ক্ষেত্রে শেষেরটা বাদে প্রত্যেকটা আইটেমই প্যাকেজ হতে হবে। আর শেষের আইটেমটা মডিউল বা প্যাকেজ যেকোন কিছু হতে পারবে। কিন্তু আগের আইটেমে ডিফাইন করা ক্লাস, ফাংশন বা ভ্যারিয়েবল হতে পারবে না।

আচ্ছা, যদি আমরা from sound.effects import \* এভাবে ইমপোর্ট করতে চাই?

যখন প্যাকেজের ভিতরের কোন ফাইলে সেই প্যাকেজের কোন মডিউল বা ফাংশনকে আমদানি করার দরকার পড়বে তখন নরমালি ইমপোর্ট করতে পারি। আবার রিলেটিভ ইমপোর্টও ব্যবহার করতে পারি। এই প্রক্রিয়ায় vocoder.py ফাইলে sound.effects প্যাকেজের echo মডিউলকে নিচের মত করে আমদানি করতে পারি।

from . import echo

বিষয়টা জটিল হয়ে গেল মনে হচ্ছে। আরেকটা নতুন প্যাকেজ স্ট্রাকচার বিবেচনা করি।

my\_package/

my\_package/\_\_init\_\_.py

my\_package/main.py

my\_package/name.py

my\_package/school.py

my\_package/college.py

এখানে main.py ফাইলে যদি আমরা school.py কে আমদানি করতে চাই অথবা school.py এর কোন ফাংশনকে আমদানি করতে চাই, তবে এভাবে করতে পারি:

from . import school

from .school import function1, function2

### পাইথন প্যাকেজ ইনডেক্স (PyPI)

[PyPI](https://pypi.python.org/pypi/) ( <https://pypi.python.org/pypi/> ) হল পাইথনের প্যাকেজ রিপোজিটরি (সংক্ষেপে রেপো বলে)। বর্তমানে এখানে প্রায় ৯১২৫৯ টি প্যাকেজ রয়েছে। এই সবগুলো প্যাকেজই কিন্তু থার্ড-পার্টি প্যাকেজ। মানে পাইথনের মূল ডেভেলপাররা নয়, বরং আমাদের মত তৃতীয় পক্ষ এগুলো ডেভেলপ করে। এগুলো পাইথনের সাথে ডিফল্টভাবে আসে না। আলাদাভাবে ইন্সটল করে ব্যবহার করতে হয়। এই থার্ড-পার্টি প্যাকেজগুলোই পাইথনের মূল শক্তি। এরাই পাইথনকে করে তুলেছে অনন্য, করে তুলেছে আনবিটেবল।

কথা হল, কিভাবে এই প্যাকেজগুলি আমরা ইন্সটল করতে পারি? এজন্য আমাদের দরকার পাইথন প্যাকেজ ইন্সটলার pip। উবুন্টুতে পাইথন-৩ এর জন্য pip ইন্সটল করতে টার্মিনালে নিচের কমান্ড চালাতে পারি আমরা:

sudo apt-get install python3-pip

এবার ধরলাম, রেপো থেকে requests প্যাকেজটা ইন্সটল করব আমরা। এজন্য উবুন্টু টার্মিনালে নিচের কমান্ড চালাব আমরা:

sudo pip3 install requests

শুধু উবুন্টুর মত লিনাক্স ডিস্ট্রো নয়, উইন্ডোজ, ম্যাকওএস সব প্লাটফর্মই pip সাপোর্ট করে। আমাদেরকে শুধু আমাদের প্লাটফর্মের জন্য pip ও প্যাকেজ ইন্সটল করতে হবে। যাহোক, প্যাকেজ ইন্সটল হয়ে গেলে আমরা একে নরমাল প্যাকেজের মত করেই ব্যবহার করতে পারি।

>>> import requests

>>> r = requests.get('https://api.github.com/events')

>>> r.status\_code

200

আর দিন শেষে দিনের সেরা ম্যাজিক। পাইথন শেলে কিছু লিখব আবার।

## পাইথনে ফাইলের নাম এবং মডিউল নিয়ে বিভ্রাট

আচ্ছা, একটা ফাইল ক্রিয়েট করি email.py নামে। এর ভেতর শুধুমাত্র এই দুটি লাইন লিখে সেভ করি:

**import** requests

print('I have been executed!')

এখন ফাইলটা রান করি:

কি দেখলেন! নিশ্চই পাইথন এরকম বিদঘুটে ধরনের এক্সেপশন থ্রো করেছে:

আচ্ছা এখন ফাইলটাকে রিনেম করি a.py নামে।

এখন আবার রান করি:

এখন আউটপুট এসেছে এরকম:

I have been executed!

কিন্তু কেন এ বৈষম্য!

requests মডিউলটার মধ্যে email নামের আরেকটা মডিউলকে কল করা হয়েছে, যা পাইথন স্ট্যান্ডার্ড লাইব্রেরির অংশ। এখন পাইথন নিয়ম অনুযায়ী কোন মডিউল প্রথমে করেন্ট ওয়ার্কিং ডিরেক্টরিতে খোঁজে, এরপর অন্য যায়গায়। যখন আমাদের ফাইলের নাম email.py ছিল, তখন পাইথন এটাকেই স্ট্যান্ডার্ড লাইব্রেরির email মডিউল ধরে নিয়েছে। তাই ওরকম বিদঘুটে এক্সেপশন রেইজ হয়েছে। আবার রিনেম করে a.py করার পর সব ঠিক ঠাক।

পাইথনের মডিউল লুক আপ অর্ডার দেখতে ইন্টারএক্টিভ শেল চালু করে এরকম লিখুন:

>>> import sys

>>>

>>> sys.path

সুতরাং, আমাদের পাইথন ফাইল নামকরণের সময় একটু সতর্ক থাকা লাগবে, চেষ্টা করতে হবে তা যেন অন্য কোন লাইব্রেরির নামে না হয়।

## Python standard module library

<https://docs.python.org/3/library/>

## মডিউল এবং প্যাকেজ

#### Standard Library Modules

পাইথনের স্ট্যান্ডার্ড লাইব্রেরিতে অনেক গুলো মডিউল রয়েছে। যে গুলো আমরা আমাদের প্রজেক্টে ইম্পোর্ট করতে পারি এবং মডিউলের মেথড গুলো ব্যবহার করতে পারি।

* <https://docs.python.org/3.4/library/>

এখানে গিয়ে সব গুলো লিস্ট পাওয়া যাবে।অনেক গুলো মডিউল আমরা দেখতে পাবো। এগুলোতে একবার চোখ বুলিয়ে নেওয়া যেতে পারে। কোন মডিউল দিয়ে কি করা যায়, এসব জানলেই হবে। এরপর যখন দরকার হবে আমরা রেফারেন্স দেখে আমাদের প্রজেক্টে প্রয়োগ করতে পারব।

আমরা দুই একটা উদাহরণ দেখব।

#### math — Mathematical functions

**math.ceil(x)**

ceil এর কাজ হচ্ছে একটা ফ্লোটিং পয়েন্ট এর পরের ইন্টিজার ভ্যালু দেওয়া। পরের বলতে যদি একটা ফ্লোটিং পয়েন্ট ভ্যালু হয় 6.2 এটার ceil ভ্যালু হবে 7, যদিও আমরা সাধারনত 6.2 এর কাছা কাছি ইন্টিজার হিসেব করি 6. আবার 6.9 এর ceil ভ্যালু হচ্ছে 7।

|  |  |
| --- | --- |
| 1  2 | import math  print (math.ceil(6.2 )) |

#### math.floor(x)

floor এর ক্ষেত্রে হয় ceil এর উলটো। ফ্লোট্রিং পয়েন্টের দশমিক মান যত বড়ই হোক, floor আমাদের তার আগের ইন্টিজার ভ্যালুটি আমাদের দিবে। আগের বলতে যদি একটা ফ্লোটিং পয়েন্ট ভ্যালু হয় 6.9 এটার floor ভ্যালু হবে 6, যদিও আমরা সাধারনত 6.9 এর কাছা কাছি ইন্টিজার হিসেব করি 7

|  |  |
| --- | --- |
| 1  2  3 | import math  print (math.floor(6.9 ))  math.sqrt(x) |

sqrt একটা সংখ্যার বর্গমূল বের করার জন্য ব্যবহার করা হয়ঃ

|  |  |
| --- | --- |
| 1  2 | import math  print (math.sqrt(9)) |

sin, cos, ten এসবের মান ও আমরা সহজে বের করতে পারি। যেমন

math.cos(x) রেডিয়ানের cos ভ্যালু দিবে আমাদের। cos (0) এর মান আমরা জানি ১, প্রোগ্রামটি রান করালে তাই আউটপুট পাবো আমরাঃ

|  |  |
| --- | --- |
| 1  2 | import math  print (math.cos(0 )) |

এভাবে ম্যাথ লাইব্রেরীর অন্যান্য মেথড গুলো আমরা দেখে নিতে পারি।

#### random

random লাইব্রেরী ব্যবহার করে আমরা রেন্ডম নাম্বার তৈরি করতে পারি। যেমনঃ

|  |  |
| --- | --- |
| 1  2 | import random  print ( random.random()) |

এটি আমাদের ০-১ এর মধ্যে একটা রেন্ডম নাম্বার দিবে। আমরা ইচ্ছে করলে একটা রেঞ্জ দিয়ে দিতে পারি। যার মধ্যে আমরা রেন্ডম নাম্বারটি চাইঃ

|  |  |
| --- | --- |
| 1  2 | import random  print ( random.randrange( 10)) |

এটি আমাদের ১ থেকে ১০ এর মধ্যে যে কোন একটা রেন্ডম নাম্বার দিবে।

আমরা চাইলে নির্দিষ্ট রেঞ্জের ভেতর রেন্ডম নাম্বার তৈরি করতে পারি, যেমনঃ

|  |  |
| --- | --- |
| 1  2 | import random  print ( random.randrange(20 , 100 )) |

এটি ২০-১০০ এর মধ্যে একটা রেন্ডম নাম্বার দিবে।  
পাইথন মডিউল ও PyPi

আমরা যখন বড় সড় কোন প্রজেক্টে কাজ করব, তখন প্রজেক্টের এক একটা ফিচার এক একটা ফাইলে আলাদা করে রাখব। আর এই আলদা করে রাখাটাই হচ্ছে মডিউল। মডিউল আকারে কোড লিখলে অন্য যে কোন প্রজেক্টে ঐ একই মডিউল ব্যবহার করতে পারি। আরেকটা সুবিধে হচ্ছে ডিস্টিবিউশন। আমরা চাইলে মডিউল তৈরি করে যে কারো সাথে আমাদের মডিউল শেয়ার করতে পারি।  
মডিউল আর কিছুইই না, আলাদা একটা ফাইল। ফাইলের নাম হচ্ছে মডিউলের নাম। যেমন odds.py নামে আমাদের একটা মডিউল আছে। যার মধ্যে getOdds নামে একটা মেথড রয়েছে। যেটাকে কল করলে আমাদের কিছু বিজোড় সংখ্যার লিস্ট দিবে। কল করার সময় আমরা বলে দিতে পারব কয়টা কত পর্যন্ত লিস্ট চাচ্ছি। তো এমন একটা মডিউলটা সহজেই আমরা তৈরি করে নিতে পারি। PyCharm ব্যবহার করে থাকলে প্রজেক্টের উপর রাইট ক্লিক করে New > File এ ক্লিক করব।

এরপর ফাইলের একটা নাম দিব। যেমন odds.py। এরপর এতে নিচের কোড গুলো লিখবঃ

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | def getOdds(n):      result = []      b = 1      while b < n:          result.append(b)          b +=2      return result |

অন্য যে কোন আইডিইতেও একই ভাবে ফাইল তৈরি করা যাবে। শুধু খেলার রাখতে হবে প্রজেক্ট মানে মূল ফাইলটা যে ফোল্ডারে, রয়েছে, odds.py ও একই ফোল্ডারে যেন থাকে।  
এবার আমরা আমাদের মূল প্রোগ্রামে এই মডিউলটা ইম্পোর্ট করে ব্যবহার করতে পারব এভাবেঃ

|  |  |
| --- | --- |
| 1  2 | import odds  print(odds.getOdds(10)) |

পাইথনের স্ট্যান্ডার্ড প্যাকেজের মতই আমরা ইম্পোর্ট করতে পারি। ইম্পোর্ট করতে হয় ফাইলের নাম ব্যবহার করে। odds.py হচ্ছে আমাদের ফাইলের না, এখানে .py অংশটা লিখতে হয় না।

getOdds হচ্ছে odds মডিউলের একটা মেথড। আমরা এরপর একে কল করলাম। প্যারামিটার হিসেবে পাস করেছি ১০। এটি আমাদের ১-১০ পর্যন্ত বিজোড় সংখ্যা গুলোর লিস্ট রিটার্ণ করবে। এরপর আমরা তা প্রিন্ট করেছি।

আমরা এখানে খুব সহজ একটা মডিউল তৈরি করেছি। কিন্তু আমরা শিখে গিয়েছি কিভাবে মডিউল তৈরি করা যায়। কি ভাবে মডিউলকে ব্যবহার করতে হয়। এবার আপনি চাইলে নিজের ক্রিয়েটিভিটি প্রয়োগ করে যে কোন মডিউল তৈরি করে নিতে পারেন নিজের প্রয়োজন মত করে।

#### **থার্ড পার্টি মডিউল**

পাইথনের স্ট্যান্ডার্ড লাইব্রেরীর মডিউল গুলো ছাড়াও আমরা থার্ড পার্টি মডিউল ব্যবহার করতে পারি। প্রোগ্রামাররা যে কোড গুলো লিখে সবার ব্যবহার করার জন্য উন্মুক্ত করে দিয়েছে, আমরা সে সব কোড গুলোও আমাদের প্রজেক্টে ব্যবহার করতে পারি। মডিউল হচ্ছে একটা সিঙ্গেল ফাইল, যার মধ্যে অনেক গুলো মেথড থাকে। অনেক গুলো ফাইল মিলে তৈরি হয় একটা প্যাকেজ। এক একটা প্যাকেজ এক একটা কাজে ব্যববার করা যায়। এমন থার্ড পার্টি অনেক প্যাকেজের ইন্ডেক্স হচ্ছে PyPi। সব গুলো পাওয়া যাবে <https://pypi.python.org/pypi> এ।

এখানে ভিজিট করলে দেখতে পাবো অনেক গুলো প্যাকেজ। আমাদের নিজেদের প্রজেক্টে এসব প্যাকেজের যে কোনটাই আমরা ব্যবহার করতে পারব। আমাদের জন্য দরকারী অনেক কোড আগে থেকেই লেখা রয়েছে। নিজেরা শুরু থেকে না লিখে আগের কোড ব্যবহার করলে অনেক সময়ই তো বেচে যাবে।

PyPi এর প্যাকেজ গুলো ব্যবহার করার জন্য আমাদের Pip ইন্সটল করে নিতে হবে। Pip ব্যবহার করে ঐ প্যাকেজ গুলো আমরা আমাদের প্রজেক্টের জন্য প্রথমে ডাউনলোড করব এবং প্রজেক্টে ইম্পোর্ট করে ব্যবহার করতে পারব। পাইথনের স্ট্যান্ডার্ড প্যাকেজ গুলো সেভাবে ব্যবহার করা যায়, ঠিক সেভাবে। তার জন্য <https://pip.pypa.io/en/latest/installing.html> এখানে গিয়ে get-pip.py পাইথন ফাইলটা কম্পিউটারে সেভ করতে হবে। যেমন আমরা সেভ করলাম Desktop এ।  
কমান্ডলাইনে ডিরেক্টরি পরিবর্তন করে Desktop এ এসে নিচের কমান্ডটা রান করলে Pip আমাদের কম্পিউটারে ইনস্টল হবেঃ  
python get-pip.py  
তাহলে Pip আমাদের কম্পিউটারে ইন্সটল হবে যা আমাদের জন্য setuptools ইন্সটল করে দিবে। এবার আমরা যে কোন প্যাকেজ ইন্সটল করার জন্য প্রস্তুত।

## (gap )sys মডিউল

এ অধ্যায়ে আমরা সিস্টেম-স্পেসিফিক কিছু প্যারামিটার ও ফাংশন নিয়ে আলোচনা করব। আর এসবের সাপ্লাই আসবে সিস্টেম মডিউল সংক্ষেপে sys মডিউল থেকে।

### sys.argv

সকালে উঠিয়া আমি মনে মনে বলি, পাইথন স্ক্রিপ্টের কাছে কিছু আর্গুমেন্ট পাচার করি। কমান্ডলাইনের মাধ্যমে খুব সহজেই আমরা পাইথন স্ক্রিপ্টে আর্গুমেন্ট পাস করতে পারি। এটা নিয়ে অতটা ভাবনা-চিন্তার অবকাশ নেই। কিন্তু সমস্যা হলো পাচার করা সেই আর্গুমেন্টগুলো স্ক্রিপ্টের মধ্যে পড়ব কীভাবে?

বুদ্ধিমানের জন্য ইশারাই যথেষ্ট। আমরা অবশ্যই বুঝে গেছি যে, এ জন্য আমাদের sys.argv-কে ব্যবহার করতে হবে। তবে এখানে জানার মতো একটা মজার জিনিস আছে। sys.argv আসলে কোনো ফাংশন নয়; বরং এটি একটি লিস্ট। আমাদের test.py ফাইলে একটা ছোট্ট গুলুগুলু মতো প্রোগ্রাম লেখা যাক:

import sys

print(sys.argv)

print(type(sys.argv))

for arg in sys.argv:

print(arg)

এবার আমরা টার্মিনালে কমান্ড দিয়ে এই স্ক্রিপ্টটা রান করাব। পাশাপাশি এটাতে কিছু আর্গুমেন্টও পাচার করব।

python3 test.py 1 2 3 4 5

**আউটপুট**

['test.py', '1', '2', '3', '4', '5']

<class 'list'>

test.py

1

2

3

4

5

আমরা আমাদের স্ক্রিপ্টে 1 2 3 4 5-কে আর্গুমেন্ট হিসেবে পাস করেছি। এসব আর্গুমেন্ট sys.argv লিস্টে জমা হয়েছে। সবচেয়ে মজার বিষয় হলো, এই লিস্টের ইনডেক্স 0-তে খোদ স্ক্রিপ্ট নিজেই আছে। আর আমাদের দেওয়া আর্গুমেন্টগুলো আছে ইনডেক্স 1 থেকে। আসলে sys.argv লিস্টের প্রথম আইটেম ওই স্ক্রিপ্ট নিজেই।

### sys.exc\_info()

এটা একটা ইন্টারেস্টিং ফাংশন। এই ফাংশন একটা টাপল রিটার্ন করে। যে এক্সেপশনটা বর্তমানে হ্যান্ডেল হচ্ছে, সেটা সম্পর্কিত তিনটি ইনফরমেটিভ ভ্যালু থাকে টাপলটায়। আর পুরো সিস্টেমে কোনো এক্সেপশন হ্যান্ডেল না হলে তিনটি None ভ্যালু থাকে টাপলে। যেমন :

import sys

try:

print(10/0)

except ZeroDivisionError:

print(sys.exc\_info())

**আউটপুট**

(<class 'ZeroDivisionError'>, ZeroDivisionError('division by zero',), <traceback object at 0x7fc456767588>)

ZeroDivisionError এক্সেপশন হ্যান্ডেল করার সময় আমরা sys.exc\_info () ফাংশনটার আউটপুট প্রিন্ট করে দেখেছি। হ্যাঁ, এটাই টাপল ও টাপলে তিনটি আইটেমও আছে। আর সেসব আইটেমে এক্সেপশন-সম্পর্কিত তথ্য বিদ্যমান।

#### sys.executable

এটা মূলত একটি স্ট্রিং। পাইথন ইন্টারপ্রিটারের এক্সিকিউটেবলের পাথ পাওয়া যায় এখানে (শুধু পাইথন ইন্টারপ্রিটারের প্রচারের জন্য)।

>>> import sys

>>> print(sys.executable)

/usr/bin/python3

#### sys.exit()

পাইথন থেকে বের হয়ে যাওয়ার জন্য এই ফাংশনটা ব্যবহৃত হয়। মন চাইলে একটা প্যারামিটার পাস করা যায়। না করলে কোনো ক্ষতি নেই। প্যারামিটারটি ইন্টিজার হতে পারে (সে ক্ষেত্রে ডিফল্ট ভ্যালু হলো জিরো) বা অন্য টাইপের অবজেক্টও হতে পারে। ইন্টিজারটি মূলত স্ট্যাটাস কোড। স্ট্যাটাস কোড জিরো মানে ‘সাকসেসফুল টার্মিনেশন’ এবং নন-জিরো মানে ‘অ্যাবনরমাল টার্মিনেশন’। তবে অপারেটিং সিস্টেমভেদে হয়তো কিছুটা পরিবর্তন থাকতে পারে। একটা উদাহরণ দেখা যাক :

>>> import sys

>>> sys.exit()

এবার আরেকটা উদাহরণ দেখা যাক :

>>> import sys

>>> sys.exit(1)

আউটপুট আপাতভাবে এক হলেও এর মধ্যে পার্থক্য আছে :

>>> import sys

>>> sys.exit("some error message")

some error message

এটা হলো একটা প্রোগ্রামকে নগদে জায়গায় বন্ধ করে দেওয়ার মুশকিল আসান তরিকা। আর আরেকটা ব্যাপার, আমরা যখন এই ফাংশনটি কল করি, তখন SystemExit এক্সেপশন রেইজ হয়।

#### sys.path

sys.path হলো স্ট্রিংয়ের লিস্ট যেখানে মিডিউলের সার্চপাথ থাকে। আরেকটু সহজভাবে বলতে গেলে, sys.path হলো একটা লিস্ট আর লিস্টের আইটেম হলো আমাদের কম্পিউটারের বিভিন্ন লোকেশন। যখন আমরা প্রোগ্রামে কোনো মডিউল আমদানি (ইমপোর্ট) করে কাজ করি, তখন পাইথন স্বয়ংক্রিয়ভাবে এসব লোকেশনে সার্চ করে মডিউলটাকে আমদানি করার চেষ্টা করে। আর না পারলে ImportError এক্সেপশন থ্রো করে।

>>> import sys

>>> type(sys.path)

<class 'list'>

>>> len(sys.path)

8

>>> sys.path

['', '/usr/lib/python3/dist-packages', '/usr/local/lib/python3.6/dist-packages/Mezzanine-4.2.0-py3.6.egg', '/usr/lib/python36.zip', '/usr/lib/python3.6', '/usr/lib/python3.6/plat-x86\_64-linux-gnu', '/usr/lib/python3.6/lib-dynload', '/usr/local/lib/python3.6/dist-packages']

>>> for path in sys.path:

... print(path)

...

/usr/lib/python3/dist-packages

/usr/local/lib/python3.6/dist-packages/Mezzanine-4.2.0-py3.6.egg

/usr/lib/python36.zip

/usr/lib/python3.6

/usr/lib/python3.6/plat-x86\_64-linux-gnu

/usr/lib/python3.6/lib-dynload

/usr/local/lib/python3.6/dist-packages

এখানে খেয়াল করলে দেখা যাবে sys.path[0] এ কিছু নেই, একটা ফাঁকা স্ট্রিং। এর কারণ হলো, প্রথম সার্চপাথটা হলো স্ক্রিপ্টটা যে ডিরেক্টরিতে আছে, সেই ডিরেক্টরিটা। এরপর পর্যায়ক্রমে পাইথন ইনডেক্স অনুসারে অন্যান্য লোকেশনে সার্চ করে। আমরা চাইলে সিস্টেম পাথে নতুন লোকেশন যোগ করতে পারি। উদাহরণ দেখা যাক :

>>> sys.path.append('/home/maateen')

>>> sys.path

['', '/usr/lib/python3/dist-packages', '/usr/local/lib/python3.6/dist-packages/Mezzanine-4.2.0-py3.6.egg', '/usr/lib/python35.zip', '/usr/lib/python3.6', '/usr/lib/python3.6/plat-x86\_64-linux-gnu', '/usr/lib/python3.6/lib-dynload', '/usr/local/lib/python3.6/dist-packages', '/home/maateen']

এখানে আমাদের অ্যাপেন্ড করা লোকেশনটা যোগ হয়েছে।

#### sys.platform

sys.platform এ প্ল্যাটফরম আইডেন্টিফায়ার স্ট্রিং থাকে। আসলে বর্তমানে আমরা যে অপারেটিং সিস্টেমে পাইথন চালাচ্ছি তার নামধাম পাওয়ার জন্যই এই sys.platform ব্যবহার করা হয়। অনেক সময় দেখা যায়, অপারেটিং সিস্টেম বদলের কারণে কোডের লজিকে পরিবর্তন করা লাগতে পারে। সেসব ক্ষেত্রে sys.platform ব্যবহার করে বিভিন্ন প্ল্যাটফরমের জন্য বিভিন্ন লজিক লেখা সম্ভব।

>>> import sys

>>> sys.platform

'linux'

আমি লিনাক্স ডিস্ট্রো উবুন্টু ব্যবহার করছি, তাই আমার আইডেন্টিফায়ার linux। এরকমভাবে প্রতিটি ওএসের জন্য আলাদা আলাদা আইডেন্টিফায়ার আছে।

একনজরে সবগুলো আইডেন্টিফায়ার দেখে নেওয়া যাক :

| **অপারেটিং সিস্টেম** | **প্ল্যাটফরম ভ্যালু** |
| --- | --- |
| Linux | linux |
| Windows | win32 |
| Windows/Cygwin | cygwin |
| Mac OS X | darwin |

### sys.stdin, sys.stdout ও sys.stderr

এগুলো হলো ফাইল অবজেক্ট। স্ট্যান্ডার্ড ইনপুট, আউটপুট ও এররের জন্য ইন্টারপ্রিটার এগুলো ব্যবহার করে থাকে :

* input() সহ সব ইন্টারেক্টিভ ইনপুটের ক্ষেত্রে stdin ব্যবহৃত হয়।
* print() ও এক্সপ্রেশন স্টেটমেন্টের আউটপুট এবং input()-এর প্রম্পটের (prompt) জন্য stdout ব্যবহৃত হয়।
* ইন্টারপ্রিটারের নিজস্ব প্রম্পট (prompt) ও এরর মেসেজ stderr-এর মাধ্যমে হ্যান্ডেল হয়।

অনেক সময় stdout ও stderr-এর আউটপুট কাস্টম লগে রাখার দরকার হতে পারে। এ জন্যই শিখে নেওয়া। এ বিষয়ে আরও বেশি জানার জন্য পাইথনের (<https://docs.python.org/3/library/sys.html>) অফিশিয়াল ডকুমেন্টেশনে ঘুরে আসতে পারেন।

## os মডিউল

পাইথন স্ট্যান্ডার্ড লাইব্রেরির খুব বিখ্যাত একটি মডিউল হলো অপারেটিং সিস্টেম(OS) মডিউল। সচরাচর এর ব্যবহার খুব বেশি। মূলত অপারেটিং সিস্টেমনির্ভর ফাংশনালিটির জন্য এটাকে ব্যবহার করা হয়। আমরা এবার এর কিছু ফাংশন সম্পর্কে জানব ও তাদের ব্যবহার করব।

### os.getcwd()

এই ফাংশন কারেন্ট ওয়ার্কিং ডিরেক্টরি রিটার্ন করে। অর্থাৎ বর্তমানে আমাদের স্ক্রিপ্ট যে ডিরেক্টরিতে আছে, সেটা পাওয়া যায়।

>>> import os

>>> os.getcwd()

'/home/maateen'

### os.chdir()

এই ফাংশন দিয়ে ডিরেক্টরি চেঞ্জ করা হয়। এটা অনেকটা ইউনিক্স কমান্ড cd-এর সমতুল্য। আমরা কারেন্ট ওয়ার্কিং ডিরেক্টরি পরিবর্তন করে অন্য কোনো ডিরেক্টরিতে যেতে চাইলে এই ফাংশনের মধ্যে সেই ডিরেক্টরির পাথ পাস করব। যেমন:

>>> import os

>>> os.getcwd()

'/home/maateen'

>>> os.chdir('/home/maateen/Desktop')

>>> os.getcwd()

'/home/maateen/Desktop'

### os.mkdir()

এই ফাংশনটি লিনাক্সের mkdir কমান্ডের সমতুল্য। যখন কোনো ডিরেক্টরি (ডিরেক্টরি - এভাবেও পড়া যায়) বা ফোল্ডার তৈরি করার দরকার পড়ে তখন এই ফাংশন ব্যবহার করা হয়। যে ফোল্ডার তৈরি করা হবে সেটার পাথটা শুধু পাস করতে হয়। যেমন :

>>> os.getcwd()

'/home/maateen/Desktop'

>>> os.mkdir('new\_folder')

এখন আমি /home/maateen/Desktop ডিরেক্টরিতে গেলে দেখতে পাব সেখানে new\_folder নামে নতুন একটা ফোল্ডার তৈরি হয়ে গেছে।

### os.makedirs()

এই ফাংশনটা লিনাক্সের mkdir –p কমান্ডের সমতুল্য। আমরা আগে একটা উদাহরণ দেখব। তারপর ব্যাখ্যায় যাব।

>>> os.getcwd()

'/home/maateen/Desktop'

>>> os.makedirs('1/2/3/4/5')

এখন আমি যদি আমার /home/maateen/Desktop ফোল্ডারে যাই তাহলে দেখব, সেখানে 1 নামে একটি ফোল্ডার আছে, 1-এর মধ্যে 2 নামে একটি ফোল্ডার, 2-এর মধ্যে আছে 3 নামে একটি ফোল্ডার, 3-এর মধ্যে আছে 4 নামে একটি ফোল্ডার এবং 4-এর মধ্যে আছে 5 নামে একটি ফোল্ডার। আসলে আমাদের এখানে দরকার ছিল 5 নামে যে ফোল্ডারটি সেটি তৈরি করা। কিন্তু 5 নামের ফোল্ডারটা আরও কয়েকটা ফোল্ডারের মধ্যে থাকবে, যা ইতিমধ্যে তৈরি করা নেই। সেগুলো আমরা একটা একটা করে তৈরি না করে এই পদ্ধতিতে এক ধাক্কায় কিন্তু পগার পার করে ফেললাম।

### os.remove()

এই ফাংশনটি লিনাক্সের rm কমান্ডের সমতুল্য। যখন কোনো একটা ফাইল ডিলিট করার দরকার পড়বে, তখন আমরা এই ফাংশন ব্যবহার করব।

>>> os.getcwd()

'/home/maateen/Desktop'

>>> os.remove('test.log')

আমার কারেন্ট ওয়ার্কিং ডিরেক্টরি থেকে test.log ফাইলটা রিমুভ করলাম।

### os.rmdir()

এই ফাংশনটা লিনাক্সের rm –r কমান্ডের সমতুল্য। যখন কোনো একটা ফোল্ডার ডিলিট করার দরকার পড়বে আমাদের তখন আমরা এই ফাংশন ব্যবহার করব।

>>> os.getcwd()

'/home/maateen/Desktop'

>>> os.rmdir('test')

আমার কারেন্ট ওয়ার্কিং ডিরেক্টরি থেকে test ফোল্ডারটা রিমুভ করলাম।

### os.rename()

একটা ফাইল বা ফোল্ডারকে রিনেম করার জন্য এই ফাংশন ব্যবহার করা হয়। উদাহরণ দেখা যাক:

>>> os.getcwd()

'/home/maateen/Desktop'

>>> os.rename('test.log', 'test.txt')

আমার কারেন্ট ওয়ার্কিং ডিরেক্টরির test.log ফাইলটাকে test.txt নামে রিনেম করলাম। os.rename () ফাংশনের মধ্যে প্রথম প্যারামিটার হিসেবে দিতে হয় সোর্স ফাইল বা ফোল্ডার আর দ্বিতীয় প্যারামিটার হিসেবে দিতে হয় যে নতুন নামে আমরা সেটাকে দেখতে চাই তার নাম।

### os.path

অপারেটিং সিস্টেম পাথ হলো অপারেটিং সিস্টেম মডিউলের একটা সাব-মডিউল। আমরা এখন এই সাব-মডিউলের কয়েকটি ফাংশন সম্পর্কে জানব।

#### os.path.basename()

এই ফাংশনের মধ্যে একটি পাথ পাস করা হলে সেটি শুধু ফাইলের নামটি রিটার্ন করবে।

>>> os.path.basename('/home/maateen/Desktop/test.py')

'test.py'

#### os.path.dirname()

এই ফাংশনের মধ্যে একটা পাথ পাস করা হলে পাথের ডিরেক্টরি অংশটুকু রিটার্ন করবে।

>>> os.path.dirname('/home/maateen/Desktop/test.py')

'/home/maateen/Desktop'

#### os.path.exists()

একটা ফাইল বা ফোল্ডার এক্সিস্ট করে কি না, তা চেক করে True বা False রিটার্ন করে।

>>> os.path.exists('/home/maateen/Desktop/test.py')

True

>>> os.path.exists('/home/maateen/Desktop')

True

#### os.path.isdir()

একটি ডিরেক্টরি এক্সিস্ট করে কি না, তা চেক করে True বা False রিটার্ন করে।

>>> os.path.isdir('/home/maateen/Desktop')

True

#### os.path.isfile()

একটা ফাইল এক্সিস্ট করে কি না, তা চেক করে True বা False রিটার্ন করে।

>>> os.path.isfile('/home/maateen/Desktop/test.py')

True

#### os.path.join()

এটা হলো জোড়াতালি লাগানোর ফাংশন। একটি ডিরেক্টরি পাথের সাথে ফাইলের নাম জোড়া দেওয়ার জন্য আমরা এই ফাংশনটি ব্যবহার করতে পারি। বোঝাই যাচ্ছে এই ফাংশনটি দুটি প্যারামিটার নিয়ে কাজ করে। প্রথম প্যারামিটার হলো ডিরেক্টরির পাথ এবং দ্বিতীয় প্যারামিটার হলো ফাইলের নাম।

>>> dir = os.getcwd()

>>> print(dir)

/home/maateen/Desktop

>>> os.path.join(dir, 'test.py')

'/home/maateen/Desktop/test.py'

#### os.path.split()

os.path.join () ফাংশনের সম্পূর্ণ বিপরীত ফাংশন হলো এটি। একটি ফাইলের ফুল পাথকে ভেঙে দুই টুকরা করতে এই ফাংশন ব্যবহৃত হয়। প্যরামিটার হিসেবে পাথ পাস করলে ফাংশনটি ডিরেক্টরি পাথ ও ফাইলের নাম টাপল হিসেবে রিটার্ন করে।

>>> os.path.split('/home/maateen/Desktop/test.py')

('/home/maateen/Desktop', 'test.py')

### os.system()

খুবই দরকারি একটা ফাংশন। কখনো কখনো পাইথন দিয়ে কোনো লিনাক্স বা উইন্ডোজ কমান্ড চালানো দরকার হতে পারে আমাদের। তখন আমরা এই ফাংশনটা ব্যবহার করব।

>>> os.system('uname -r')

4.4.0-49-generic

0

লিনাক্সে uname –r কমান্ড দিয়ে আমরা বর্তমান কার্নেলের ভার্সন দেখতে পারি। ঠিক একই কাজটি আমরা একটু ঘুরিয়ে করলাম।

আমাদের একটি ব্যাপার জানা দরকার। এতক্ষণ আমরা যেসব উদাহরণ দেখলাম, সেখানে পাথ দেখানো হয়েছে লিনাক্সের সাপেক্ষে। লিনাক্সে পাথ সেপারেটর হিসেবে ব্যবহার করা হয় স্লাশ চিহ্ন (/) কিন্তু উইন্ডোজে হয় ব্যাকস্লাশ (), যেমন— C:\Users\maateen\Documents\test.py

এই চ্যাপ্টারে আমরা সচরাচর ব্যবহৃত হওয়া অপারেটিং সিস্টেম(os) মডিউলের এ রকম ফাংশনগুলোই কেবল দেখেছি। এই চ্যাপ্টারে os.popen() ও os.spawn\*() ফাংশনগুলো নিয়ে আমরা আলোচনা করলাম না। কারণ, নতুন প্রসেস চালু করার জন্য বা আউটপুট ক্যাপচার করার জন্য অথবা রিটার্নকোড ক্যাচ করার জন্য subprocess মডিউল আরও ভালো ফ্যাসিলিটি দেয়। তাই এসব কাজের জন্য subprocess মডিউল চ্যাপ্টারে আমরা বিস্তারিত আলোচনা করব। তবে os মডিউল সম্পর্কে আরও জানতে পাইথনের অফিশিয়াল ডকুমেন্টেশনে ঘুরে আসতে পারেন - (<https://docs.python.org/3/library/os.html>)।

## subprocess মডিউল

অনেক সময় পাইথন প্রোগ্রাম থেকেই নতুন একটি প্রোগ্রাম বা প্রসেস চালু করার দরকার পড়ে অথবা একটা শেল কমান্ড চালিয়ে তার আউটপুট জানার দরকার পড়ে। সাবপ্রসেস (subprocess) মডিউলের সহযোগিতায় এসব কাজ আমরা একেবারে নির্বিঘ্নে করতে পারি। আরেকটু বিস্তারিতভাবে বলতে গেলে, সাবপ্রসেস মডিউলের সহায়তায় আমরা একটা প্রসেস শুরু করতে পারি, এর ইনপুট/আউটপুট/এরর পাইপের সাথে সংযোগ স্থাপন করতে পারি আর এর রিটার্ন কোডগুলো সহজেই দেখতে পারি। অপারেটিং সিস্টেম মডিউল চ্যাপ্টারে অবশ্য আমরা প্রোগ্রাম বা প্রসেস চালু করার একটা তরিকা শিখেছি কিন্তু সেটা একেবারে বাচ্চা পদ্ধতি। শুধুই কমান্ড চালানো যায় কিন্তু কমান্ড চালিয়ে কী আউটপুট পেলাম তা ক্যাপচার করা যায় না, একটি কমান্ড চালিয়ে পরবর্তী সময়ে ওই কমান্ডের সাথে যোগাযোগ করে ইনপুট পাস করা যায় না, এমনটি এররও ক্যাচ করা যায় না। সাবপ্রসেস মডিউল হলো এসব সমস্যার শাহি সমাধান।

### run()

রান ফাংশন আলোচনার আগে একটা তথ্য জানানোর দরকার আছে। এই ফাংশনটা পাইথন ৩.৫-এ প্রথম ইন্ট্রোডিউস করা হয়। সুতরাং এর আগের ভার্সন যদি কেউ ব্যবহার করে থাকি, তাহলে এই ফাংশনের কাজকর্ম করতে পারব না। যাহোক, পাইথন থেকে একটা প্রসেস বা শেল কমান্ড চালু করার সবচেয়ে সহজ উপায় হলো রান ফাংশনের সহায়তা নেওয়া। যেমন—

>>> import subprocess

>>> output = subprocess.run(["ls", "-l"])

total 72

drwxr-xr-x 2 maateen maateen 4096 Dec 2415:48 Desktop

drwxr-xr-x 6 maateen maateen 4096 Dec 1223:42 Documents

drwxr-xr-x 18 maateen maateen 12288 Dec 2413:54 Downloads

drwxrwxr-x 5 maateen maateen 4096 Dec 412:37 MEGAsync

drwxr-xr-x 2 maateen maateen 4096 Jul 2123:19 Music

drwxr-xr-x 10 maateen maateen 20480 Dec 2316:54 Pictures

drwxr-xr-x 3 maateen maateen 4096 Jun 222016 Public

drwxr-xr-x 2 maateen maateen 4096 Apr 262016 Templates

drwxr-xr-x 5 maateen maateen 4096 Dec 2401:26 Videos

drwxrwxr-x 16 maateen maateen 4096 Dec 1420:18 workspace

>>> print(output)

CompletedProcess(args=['ls', '-l'], returncode=0)

কী সুন্দর আউটপুট পেয়ে গেলাম! কিন্তু এত কিছুর মাঝেও একটি ঝামেলা আছে। output ভেরিয়েবলে কিন্তু আমরা আউটপুট ক্যাপচার করতে পারিনি। আসলে এভাবে হ্যাঁচকামারা কায়দায় শেল কমান্ড রান করলে কোনো আউটপুটই ক্যাপচার করা যাবে না। তাহলে কী উপায়? কী উপায়?

>>> import subprocess

>>> output = subprocess.run(['ls', '-l'], stdout=subprocess.PIPE)

>>> type(output)

<class 'subprocess.CompletedProcess'>

>>> print(output)

CompletedProcess(args=['ls', '-l'], returncode=0, stdout=b'total 30352\n-rw-rw-r-- 1 maateen maateen 93074 Nov 17 14:13 20161116092156306283.pdf\n-rw-rw-r-- 1 maateen maateen 131 Nov 20 12:21 hijibiji.py\n-rwxrwxrwx 1 maateen maateen 30966208 Oct 25 19:35 python\_101.pdf\n-rw-rw-r-- 1 maateen maateen 95 Dec 24 21:37 test.py\n')

>>> type(output.stdout)

<class 'bytes'>

>>> result = output.stdout.decode('UTF-8')

>>> print(result)

total 30352

-rw-rw-r--1 maateen maateen 93074 Nov 1714:1320161116092156306283.pdf

-rw-rw-r--1 maateen maateen 131 Nov 2012:21 hijibiji.py

-rwxrwxrwx 1 maateen maateen 30966208 Oct 2519:35 python\_101.pdf

-rw-rw-r--1 maateen maateen 95 Dec 2421:37 test.py

এটা কিন্তু একটা উপায় বটে! রান ফাংশনে আমরা নতুন একটা প্যারামিটার stdout=subprocess.PIPE পাস করেছি। তারপর যে আউটপুটটা পেলাম সেটা আউটপুট ভেরিয়েবলে ক্যাপচার করে আউটপুটের টাইপ চেক করেছি। এখানে আছে সাবপ্রসেস মডিউলের CompletedProcess ক্লাসের একটি অবজেক্ট। তারপর এর মধ্যে কী আছে দেখার জন্য আমরা ভেরিয়েবলটাকে প্রিন্ট করেছি। এখানে অনেক কিছুই আছে কিন্তু আমাদের দরকার শুধু stdout-এর উপাদানগুলো। সে জন্য আমরা output.stdout-এর টাইপ চেক করে দেখলাম, এটি বাইট টাইপের। বাইট তো কী হয়েছে? আমরা তো চাইলেই এটিকে UTF-8-এ ডিকোড করতে পারি। এবং ডিকোড করার পর আমরা পেয়ে গেলাম আমাদের কাঙ্ক্ষিত রেজাল্ট।

একটা কথা বলে রাখা ভালো যে— subprocess.run(args, \*, stdin=None, input=None, stdout=None, stderr=None, shell=False, timeout=None, check=False, encoding=None, errors=None) সবসময় CompletedProcess ক্লাসের একটা ইনস্ট্যান্স রিটার্ন করে। encoding আর errors প্যারামিটার দুটি পাইথন ৩.৬-এ ইন্ট্রোডিউস করা হয়েছে।

আর রান ফাংশনের মধ্যে আমরা প্রথম প্যারামিটার হিসেবে সব সময় প্রসেস লঞ্চ (শুরু) করার কমান্ড বা শেল কমান্ড দেব। কিন্তু এখানে চিন্তাভাবনার অবকাশ আছে। একটা উদাহরণ দিলে পরিষ্কার হয়ে যাবে:

>>> import subprocess

>>> output = subprocess.run('uname')

Linux

>>> output = subprocess.run('uname -r')

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

File "/usr/lib/python3.6/subprocess.py", line 693, in run

with Popen(\*popenargs, \*\*kwargs) as process:

File "/usr/lib/python3.6/subprocess.py", line 947, in \_\_init\_\_

restore\_signals, start\_new\_session)

File "/usr/lib/python3.6/subprocess.py", line 1551, in \_execute\_child

raise child\_exception\_type(errno\_num, err\_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'uname -r'

>>> output = subprocess.run(['uname', '-r'])

4.4.0-58-generic

আমরা যদি শুধু সিঙ্গেল ওয়ার্ডেড কমান্ডকে স্ট্রিং আর্গুমেন্ট হিসেবে পাস করি তাহলে কোনো ঝামেলা নেই। কিন্তু যদি এমন কোনো কমান্ড দিতে চাই যার মধ্যে স্পেস রয়েছে, সে ক্ষেত্রে লিস্ট আকারে আর্গুমেন্ট পাস করতে হবে। যেমনটা আমরা আগের প্রোগ্রামে শেষের দিকে করেছি। এটাকে বলে টোকেনাইজেশন। আর একটু পরেই আমরা এর সম্পর্কে আরেকটু জানব।

### Popen ক্লাস

সত্যি কথা বলতে কি, Popen ক্লাসের মাধ্যমেই সাবপ্রসেস মডিউলে যাবতীয় প্রসেস লঞ্চ (শুরু) ও ম্যানেজ করা হয়। এটার সবচেয়ে বড় সুবিধা হলো, এটা খুবই ফ্লেক্সিবল। তাই ডেভেলপারদের কম জ্বালা সহ্য করতে হয়।

class subprocess.Popen(args, bufsize=-1, executable=None, stdin=None, stdout=None, stderr=None, preexec\_fn=None, close\_fds=True, shell=False, cwd=None, env=None, universal\_newlines=False, startupinfo=None, creationflags=0, restore\_signals=True, start\_new\_session=False, pass\_fds=(), \*, encoding=None, errors=None) এ অনেকগুলো প্যারামিটার রয়েছে। একটা উদাহরণ দেখা যাক :

import subprocess

output = subprocess.Popen('uname')

print(type(output))

print(output)

**আউটপুট**

<class 'subprocess.Popen'>

<subprocess.Popen object at 0x7f585724a748>

Linux

সবকিছু কিন্তু অনেকটা ওই আগের run() ফাংশনের মতোই। পার্থক্য শুধু একটাই যে আউটপুট ক্যাপচার করার জন্য এক্সট্রা কোনো কারুকাজ করা লাগেনি। আর আর্গুমেন্ট পাসিংয়ের ক্ষেত্রে run() ফাংশনের নিয়মাবলিই প্রযোজ্য। কিন্তু কমান্ড টোকেনাইজেশনের জন্য আমরা একটা নতুন পদ্ধতি শিখব। আর সে জন্য ব্যবহার করব shlex মডিউলের split() ফাংশনকে।

import shlex, subprocess

command = 'uname -r'

args = shlex.split(command)

print(args)

output = subprocess.Popen(args)

print(output)

**আউটপুট**

['uname', '-r']

<subprocess.Popen object at 0x7f09acd75780>

4.4.0-58-generic

কী চমৎকার! split() ফাংশনের মধ্যে আমরা আমাদের কমান্ডটা পাস করলাম আর সে আমাদের টোকেনাইজড কমান্ডটা রিটার্ন করল। খুব সহজেই কিন্তু আমাদের কাজটা এবার হয়ে গেল। এবার আরেকটা উদাহরণ দেখা যাক :

import shlex, subprocess

command = 'uname -r'

args = shlex.split(command)

print(args)

output = subprocess.Popen(args)

print(output)

code = output.wait()

print(code)

**আউটপুট**

['uname', '-r']

<subprocess.Popen object at 0x7fa3ce8a4780>

4.4.0-58-generic

0

আমরা এই প্রসেসের রিটার্ন কোড চেক করে দেখেছি। একটা প্রসেসের রিটার্নকোড 0 হওয়া মানে সাকসেস আর নন-জিরো হওয়া মানে এরর। রিটার্নকোডকে এক্সিট স্ট্যাটাসও বলা হয়। ইতিপূর্বে কিন্তু run() ব্যবহার করার সময়েও আমরা returncode প্যারামিটারটা দেখেছিলাম। আচ্ছা, ক্লাসের ওই জঞ্জাল থেকে কি আমরা returncode-কে উদ্ধার করতে পারব? কীভাবে করা যায়? ভাবতে হবে! কাজটি দেখা যাক:

>>> import subprocess

>>> output = subprocess.run(["ls", "-l"])

total 72

drwxr-xr-x 2 maateen maateen 4096 Dec 2510:59 Desktop

drwxr-xr-x 6 maateen maateen 4096 Dec 1223:42 Documents

drwxr-xr-x 18 maateen maateen 12288 Dec 2509:16 Downloads

drwxrwxr-x 5 maateen maateen 4096 Dec 412:37 MEGAsync

drwxr-xr-x 2 maateen maateen 4096 Jul 2123:19 Music

drwxr-xr-x 10 maateen maateen 20480 Dec 2510:18 Pictures

drwxr-xr-x 3 maateen maateen 4096 Jun 222016 Public

drwxr-xr-x 2 maateen maateen 4096 Apr 262016 Templates

drwxr-xr-x 5 maateen maateen 4096 Dec 2401:26 Videos

drwxrwxr-x 16 maateen maateen 4096 Dec 1420:18 workspace

>>> print(output)

CompletedProcess(args=['ls', '-l'], returncode=0)

>>> print(output.returncode)

0

>>> print(type(output.returncode))

<class 'int'>

এখন Popen.communicate(input=None, timeout=None) নিয়ে আলোচনা চালানো যাক। আমরা একটা প্রসেস চালানোর পর যদি ওই প্রসেসের সাথে ইন্টারঅ্যাক্ট করার দরকার হয় তখন এই ফাংশনটা ব্যবহার করা হয়। communicate() ফাংশন আউটপুট ও এররের টাপল রিটার্ন করে। অপশনাল প্যারামিটার ইনপুটের মাধ্যমে চাইল্ড প্রসেসের কাছে ডেটা পাঠানো যায়। তবে আমরা যদি প্রসেসের stdin-এ ডেটা পাঠাতে চাই সে ক্ষেত্রে Popen অবজেক্ট stdin=PIPE-সহ তৈরি করতে হবে। একই রেজাল্ট টাপলে কারিশমা করার জন্য stdout=PIPE ও stderr=PIPE ব্যবহার করতে হবে।

পাইথনের অফিশিয়াল ডক থেকে একটা উদাহরণ দেখা যাক:

proc = subprocess.Popen(...)

try:

outs, errs = proc.communicate(timeout=15)

except TimeoutExpired:

proc.kill()

outs, errs = proc.communicate()

এখানে timeout=15 দিয়ে ১৫ সেকেন্ড পর টাইমআউট হবে বুঝিয়েছে। যদি কোনো কারণে ১৫ সেকেন্ডের মধ্যে প্রসেসটা টার্মিনেট না করে, সে ক্ষেত্রে TimeoutExpired এক্সেপশন রেইজ হবে। তবে এই এক্সেপশনটাকে হ্যান্ডেল করে আবার প্রসেসের সাথে কমিউনিকেট করা হলে কোনো আউটপুটই খোয়া যাবে না। কী শান্তি! তবে যে প্রসেস চালু করেছি তা বন্ধ করা অনিবার্য। সে জন্য আমরা kill() মেথডের সহায়তা নিতে পারি।

### call()

সাবপ্রসেস মডিউলের তিনটি আদিম ফাংশন আছে : call(), check\_call() ও check\_output()। ভবিষ্যতে এরা থাকবে কি না, তা নিয়ে জাতি সন্দিহান। run() ফাংশন দিয়ে আমরা এদের মোটামুটিভাবে রিপ্লেস করতে পারলেও কিন্তু এদের দরকার আছে।

কল ফাংশনটা খুবই সিম্পল। এর মধ্যে কমান্ড আর্গুমেন্ট হিসেবে পাস করা হলে প্রথমে কমান্ডটা চালাবে, তারপর কমান্ড চলা শেষ হবার জন্য ওয়েট করবে, শেষে returncode রিটার্ন করবে। উদাহরণ দেখা যাক:

>>> import subprocess

>>> output = subprocess.call(['ls', '-l'])

total 72

drwxr-xr-x 2 maateen maateen 4096 Dec 2510:59 Desktop

drwxr-xr-x 6 maateen maateen 4096 Dec 1223:42 Documents

drwxr-xr-x 18 maateen maateen 12288 Dec 2509:16 Downloads

drwxrwxr-x 5 maateen maateen 4096 Dec 412:37 MEGAsync

drwxr-xr-x 2 maateen maateen 4096 Jul 2123:19 Music

drwxr-xr-x 10 maateen maateen 20480 Dec 2510:18 Pictures

drwxr-xr-x 3 maateen maateen 4096 Jun 222016 Public

drwxr-xr-x 2 maateen maateen 4096 Apr 262016 Templates

drwxr-xr-x 5 maateen maateen 4096 Dec 2401:26 Videos

drwxrwxr-x 16 maateen maateen 4096 Dec 1420:18 workspace

>>> print(output)

0

আউটপুটের মধ্যে শুধু রিটার্নকোড বাদে আর কিছু নেই। আমরা চাইলে run() ফাংশন ব্যবহার করে এই প্রোগ্রামটার একটি ইকুইভ্যালেন্ট (সমতুল্য) প্রোগ্রাম লিখতে পারি।

>>> import subprocess

>>> output = subprocess.run(['ls', '-l']).returncode

total 72

drwxr-xr-x 2 maateen maateen 4096 Dec 2510:59 Desktop

drwxr-xr-x 6 maateen maateen 4096 Dec 1223:42 Documents

drwxr-xr-x 18 maateen maateen 12288 Dec 2509:16 Downloads

drwxrwxr-x 5 maateen maateen 4096 Dec 412:37 MEGAsync

drwxr-xr-x 2 maateen maateen 4096 Jul 2123:19 Music

drwxr-xr-x 10 maateen maateen 20480 Dec 2510:18 Pictures

drwxr-xr-x 3 maateen maateen 4096 Jun 222016 Public

drwxr-xr-x 2 maateen maateen 4096 Apr 262016 Templates

drwxr-xr-x 5 maateen maateen 4096 Dec 2401:26 Videos

drwxrwxr-x 16 maateen maateen 4096 Dec 1420:18 workspace

>>> print(output)

0

আরেকটি উদাহরণ দেখা যাক:

>>> subprocess.call('ls -l')

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

File "/usr/lib/python3.6/subprocess.py", line 557, in call

with Popen(\*popenargs, \*\*kwargs) as p:

File "/usr/lib/python3.6/subprocess.py", line 947, in \_\_init\_\_

restore\_signals, start\_new\_session)

File "/usr/lib/python3.6/subprocess.py", line 1551, in \_execute\_child

raise child\_exception\_type(errno\_num, err\_msg)

FileNotFoundError: [Errno 2] No such file or directory: 'ls -l'

এরর থ্রো করেছে। কারণটা আমরা আগেই জেনেছি। কমান্ড আর্গুমেন্ট টোকেনাইজড করা নয়।

>>> subprocess.call('ls -l', shell=True)

total 72

drwxr-xr-x 2 maateen maateen 4096 Dec 2510:59 Desktop

drwxr-xr-x 6 maateen maateen 4096 Dec 1223:42 Documents

drwxr-xr-x 18 maateen maateen 12288 Dec 2509:16 Downloads

drwxrwxr-x 5 maateen maateen 4096 Dec 412:37 MEGAsync

drwxr-xr-x 2 maateen maateen 4096 Jul 2123:19 Music

drwxr-xr-x 10 maateen maateen 20480 Dec 2510:18 Pictures

drwxr-xr-x 3 maateen maateen 4096 Jun 222016 Public

drwxr-xr-x 2 maateen maateen 4096 Apr 262016 Templates

drwxr-xr-x 5 maateen maateen 4096 Dec 2401:26 Videos

drwxrwxr-x 16 maateen maateen 4096 Dec 1420:18 workspace

0

এখানে কোনো এরর নেই, এবারও কিন্তু আমরা কমান্ড আর্গুমেন্ট টোকেনাইজড করে পাস করিনি। তাহলে? আসলে এটা shell=True প্যারামিটারের কারিশমা। এই প্যারামিটার True সেট করা হলে আর্গুমেন্ট হিসেবে আমরা যে কমান্ডই দেব না কেন তা শেল প্রোগ্রাম প্রসেস করবে। লিনাক্সের ক্ষেত্রে ওপরের প্রোগ্রাম হলো /bin/sh -c 'ls -l'-এর সমতুল্য। যেমন:

>>> subprocess.call(['ls', '-l'], shell=True)

Desktop Downloads Music Public Videos

Documents MEGAsync Pictures Templates workspace

0

এখানে আউটপুট পরিবর্তিত হয়ে গেছে। এখানেও ঘটনা আছে। যখন আমরা shell=True সেট করে কমান্ড আর্গুমেন্টের লিস্ট পাস করি তখন শেলের কাছে এই লিস্ট কমান্ডলাইন আর্গুমেন্ট হিসেবে পাস হয়। এ ক্ষেত্রে ওপরের প্রোগ্রামের সমতুল্য শেল কমান্ড হলো /bin/sh -c ls -l। /bin/sh সব সময় -c এর পরের আর্গুমেন্টটাকে এক্সিকিউট করে। তাই দিন শেষে শেল কমান্ডটা এ রকম হয়— /bin/sh -c ls

### check\_call()

এর মধ্যে কমান্ড আর্গুমেন্ট হিসেবে পাস করা হলে প্রথমে কমান্ডটা চালাবে, তারপর কমান্ড চলা শেষ হওয়ার জন্য ওয়েট করবে, শেষে returncode জিরো হলে জিরো রিটার্ন করবে। আর নন-জিরো হলে CalledProcessError এক্সেপশন রেইজ করবে। CalledProcessError অবজেক্টের returncode অ্যাট্রিবিউটে রিটার্ন কোড থাকবে।

>>> import subprocess

>>> output = subprocess.check\_call(['ls', '-l'])

total 72

drwxr-xr-x 2 maateen maateen 4096 Dec 2510:59 Desktop

drwxr-xr-x 6 maateen maateen 4096 Dec 1223:42 Documents

drwxr-xr-x 18 maateen maateen 12288 Dec 2509:16 Downloads

drwxrwxr-x 5 maateen maateen 4096 Dec 412:37 MEGAsync

drwxr-xr-x 2 maateen maateen 4096 Jul 2123:19 Music

drwxr-xr-x 10 maateen maateen 20480 Dec 2510:18 Pictures

drwxr-xr-x 3 maateen maateen 4096 Jun 222016 Public

drwxr-xr-x 2 maateen maateen 4096 Apr 262016 Templates

drwxr-xr-x 5 maateen maateen 4096 Dec 2401:26 Videos

drwxrwxr-x 16 maateen maateen 4096 Dec 1420:18 workspace

>>> print(output)

0

এখানে রিটার্নকোড ছিল জিরো তাই জিরো রিটার্ন করেছে, অন্যথায় এক্সেপশন রেইজ করত। আমরা চাইলে run() ফাংশন ব্যবহার করে এই প্রোগ্রামটির একটা ইকুইভ্যালেন্ট (সমতুল্য) প্রোগ্রাম লিখতে পারি।

>>> import subprocess

>>> output = subprocess.run(['ls', '-l'], check=True)

total 72

drwxr-xr-x 3 maateen maateen 4096 Dec 2512:26 Desktop

drwxr-xr-x 6 maateen maateen 4096 Dec 1223:42 Documents

drwxr-xr-x 18 maateen maateen 12288 Dec 2512:36 Downloads

drwxrwxr-x 5 maateen maateen 4096 Dec 412:37 MEGAsync

drwxr-xr-x 2 maateen maateen 4096 Jul 2123:19 Music

drwxr-xr-x 10 maateen maateen 20480 Dec 2512:26 Pictures

drwxr-xr-x 3 maateen maateen 4096 Jun 222016 Public

drwxr-xr-x 2 maateen maateen 4096 Apr 262016 Templates

drwxr-xr-x 5 maateen maateen 4096 Dec 2401:26 Videos

drwxrwxr-x 16 maateen maateen 4096 Dec 1420:18 workspace

>>> print(output)

CompletedProcess(args=['ls', '-l'], returncode=0)

### check\_output()

এর মধ্যে কমান্ড আর্গুমেন্ট হিসেবে পাস করা হলে প্রথমে কমান্ডটা চালাবে, তারপর কমান্ড চলা শেষ হওয়ার জন্য ওয়েট করবে, শেষে রিটার্নকোড জিরো হলে আউটপুট রিটার্ন করবে। আর নন-জিরো হলে CalledProcessError এক্সেপশন রেইজ করবে। CalledProcessError অবজেক্টের returncode অ্যাট্রিবিউটে রিটার্ন কোড আর output অ্যাট্রিবিউটে আউটপুট থাকবে।

>>> import subprocess

>>> output = subprocess.run(['ls', '-l'], check=True)

total 72

drwxr-xr-x 3 maateen maateen 4096 Dec 2512:26 Desktop

drwxr-xr-x 6 maateen maateen 4096 Dec 1223:42 Documents

drwxr-xr-x 18 maateen maateen 12288 Dec 2512:36 Downloads

drwxrwxr-x 5 maateen maateen 4096 Dec 412:37 MEGAsync

drwxr-xr-x 2 maateen maateen 4096 Jul 2123:19 Music

drwxr-xr-x 10 maateen maateen 20480 Dec 2512:26 Pictures

drwxr-xr-x 3 maateen maateen 4096 Jun 222016 Public

drwxr-xr-x 2 maateen maateen 4096 Apr 262016 Templates

drwxr-xr-x 5 maateen maateen 4096 Dec 2401:26 Videos

drwxrwxr-x 16 maateen maateen 4096 Dec 1420:18 workspace

>>> print(output)

CompletedProcess(args=['ls', '-l'], returncode=0)

>>> import subprocess

>>> output = subprocess.check\_output(['ls'])

>>> print(output)

b'Desktop\nDocuments\nDownloads\nMEGAsync\nMusic\nPictures\nPublic\nTemplates\nVideos\nworkspace\n'

রিটার্নকোড জিরো তাই আউটপুট রিটার্ন করেছে, অন্যথায় এক্সেপশন রেইজ করত। আমরা চাইলে run() ফাংশন ব্যবহার করে এই প্রোগ্রামটির একটি ইকুইভ্যালেন্ট (সমতুল্য) প্রোগ্রাম লিখতে পারি। যেমন:

>>> import subprocess

>>> output = subprocess.run(['ls'], check=True, stdout=subprocess.PIPE).stdout

>>> print(output)

b'Desktop\nDocuments\nDownloads\nMEGAsync\nMusic\nPictures\nPublic\nTemplates\nVideos\nworkspace\n'

call() ও check\_call() ফাংশন দুটিতে stdout=PIPE ও stderr=PIPE ব্যবহার করার ক্ষেত্রে ডেডলক সমস্যার উদগিরণ হলো। সহজ কথায়, ডেডলক হলো এমন একটি পরিস্থিতি যখন প্রসেস-১ রিসোর্স-১ ধরে বসে আছে আর রিসোর্স-২-এর জন্য অপেক্ষা করছে এবং প্রসেস-২ রিসোর্স-২ ধরে বসে আছে আর রিসোর্স-১-এর জন্য অপেক্ষা করছে। যেহেতু একটি রিসোর্স একই সময়ে কেবল একটি প্রসেসকেই অ্যালোকেট করা যায় তাই প্রসেস-১ ও প্রসেস-২ হুদাই বসে থাকবে, কোনো কাজই করতে পারবে না। সাবপ্রসেস মডিউল সম্পর্কে আরও বিস্তারিত জানার জন্য পাইথনের (<https://docs.python.org/3/library/subprocess.html>) এই অফিশিয়াল ডকুমেন্টেশন থেকে ঘুরে আসতে পারেন।

## datetime ও time মডিউল

“সময় ও নদীর স্রোত কারও জন্য অপেক্ষা করে না” - শিশুকাল থেকে এই বাক্য শুনতে শুনতে আমরা ক্লান্ত। কিন্তু তারপরও সময়ের দাম দিতে পেরেছি কজন? যারা পেরেছি তারা এখন উন্নতির চরম শিখরে অবস্থান করছি আর যারা পারিনি তারা তেলাপোকার মতো বেঁচে আছি। “অতিকায় হস্তী লোপ পাইয়াছে, কিন্তু তেলাপোকা টিকিয়া আছে।” টিকিয়া আছে তো থাকুক, আমাদের কী? আমাদের টপিক তেলাপোকার জন্মরহস্যের ওপরে নয়। আমরা শিখব পাইথনে কীভাবে তারিখ ও সময়কে হ্যান্ডেল করা যায় তার উপায়।

### datetime মডিউল

তারিখ ও সময় নিয়ে খেলাধুলার যাবতীয় ক্লাসের ভান্ডার হলো ডেটটাইম মডিউল। এই মডিউলে আমরা দুই ধরনের ডেট ও টাইম অবজেক্ট নিয়ে কথা বলব। একটি হলো aware (অ্যাওয়ার) এবং অন্যটি হলো naive (নাইভ)। অ্যাওয়ার অবজেক্টে টাইমজোন ও ডে-লাইট সেভিং সম্পর্কে পর্যাপ্ত তথ্য থাকে। ফলে সহজেই অন্য অ্যাওয়ার অবজেক্টের সাথে সম্পর্কিত করা যায়। কিন্তু নাইভ অবজেক্টকে অন্যান্য ডেট ও টাইস অবজেক্টের সাথে সম্পর্কিত করা যায় না।

আসলে মডিউলের বিস্তার ব্যাপক, শুধু এর ওপরই ছোটখাটো একটি বই লিখে ফেলা যায়। কিন্তু তারপরও আমরা গুরুত্বপূর্ণ অংশগুলোর ওপর আলোকপাত করার চেষ্টা করব।

#### datetime.date()

এই ক্লাসের প্যারামিটার তিনটি - year, month, day। সবগুলো প্যারামিটার ইন্টিজার টাইপের। সবগুলো প্যারামিটার যথাযথভাবে পাস করলে একটি date অবজেক্ট পাওয়া যায়। date অবজেক্ট তৈরি করার পর, ক্লাসের বিভিন্ন মেথডের সাহায্যে জাতিকে পাইথনিক ম্যাজিক দেখানো যায়।

>>> from datetime import date

>>> date(2018, 2, 9)

datetime.date(2018, 2, 9)

date.today() মেথড আজকের তারিখ (আঞ্চলিক সময়) রিটার্ন করে।

>>> from datetime import date

>>> date.today()

datetime.date(2018, 2, 9)

date.today() মেথড ব্যবহার করে পাওয়া বা যেকোনো date অবজেক্টকে আমরা কয়েকভাবে অ্যাক্সেস করতে পারি।

>>> from datetime import date

>>> a = date.today()

>>> a.year

2018

>>> from datetime import date

>>> my\_date = date.today()

>>> print(my\_date.year)

2018

>>> print(my\_date.month)

2

>>> print(my\_date.day)

9

আবার কোনো date অবজেক্টের অ্যাট্রিবিউটের ভ্যালু চেঞ্জ করতে চাইলে date.replace(year=self.year, month=self.month, day=self.day) মেথড ব্যবহার করতে পারি।

>>> from datetime import date

>>> my\_date = date.today()

>>> my\_date

datetime.date(2018, 2, 9)

>>> my\_date.replace(1996, 10, 23)

datetime.date(1996, 10, 23)

>>> my\_date

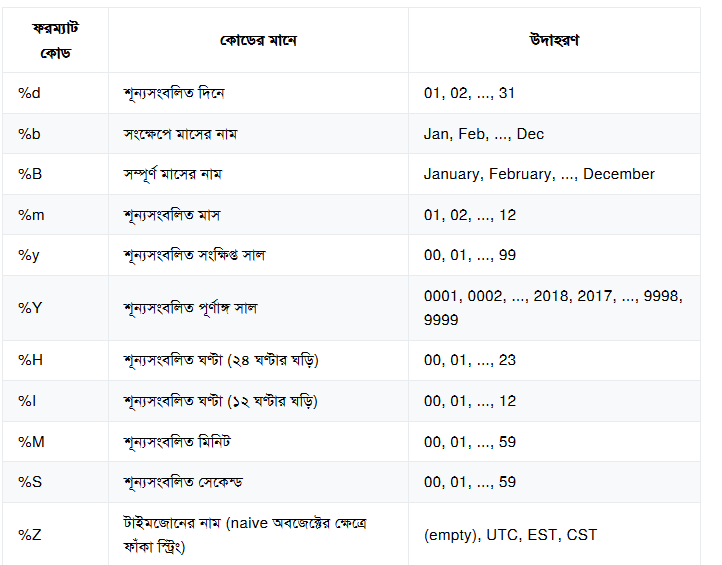
datetime.date(2018, 2, 9)

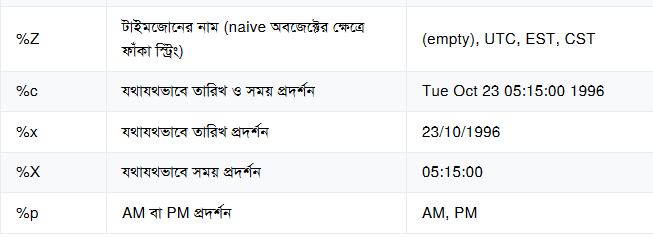
>>> my\_date = my\_date.replace(1996, 10, 23)

>>> my\_date

datetime.date(1996, 10, 23)

স্ট্রিং ফরম্যাটিংয়ের মতো করে ডেটও ফরম্যাট করা যায়। এ জন্য date.strftime(format) মেথড ব্যবহার করা হয়। এই মেথড ফরম্যাট কোডকে আর্গুমেন্ট হিসেবে নেয়। প্রথমেই আমরা প্রচলিত ফরম্যাট কোডগুলোকে দেখে নেব একনজর।





এখন আমরা প্রচলিত ফরম্যাট কোডগুলো জানি। তাহলে একটি প্রোগ্রাম লিখে ফেলা যাক!

>>> from datetime import date

>>> my\_date = date.today()

>>> my\_date

datetime.date(2018, 2, 9)

>>> my\_date.strftime('%d %B, %Y')

'09 February, 2018'

কী সুন্দরভাবে ফরম্যাট করে ফেললাম! শুধু একটা জিনিস মনে রাখতে হবে, ফরম্যাট কোডগুলোকে স্ট্রিংয়ের মতো করে পাস করতে হবে।

### datetime.time()

এই ক্লাসের কয়েকটি প্যারামিটার আছে - hour=0, minute=0, second=0, microsecond=0, tzinfo=None, \*, fold=0। সবগুলো প্যারামিটারই অপশনাল আর ইন্টিজার। ঠিকভাবে প্যারামিটার পাস করলে একটা time অবজেক্ট পাওয়া যায়। অবজেক্ট তো পেলাম কিন্তু সেই অবজেক্ট নিয়ে খেলা করব কীভাবে? এবার সেটাই আমরা শিখব।

>>> from datetime import time

>>> time(5, 15, 00)

datetime.time(5, 15)

এই time অবজেক্টকে আমরা কয়েকভাবে অ্যাক্সেস করতে পারি।

>>> from datetime import time

>>> t = time(5, 15, 00)

>>> t.hour

5

>>> t.minute

15

>>> t.second

0

>>> t.microsecond

0

টাইম অবজেক্টের বিভিন্ন অ্যাট্রিবিউটের ভ্যালু চেঞ্জ করার জন্য আমরা কিছু মেথড ব্যবহার করতে পারি। যেমন:

time.replace(hour=self.hour, minute=self.minute, second=self.second, microsecond=self.microsecond, tzinfo=self.tzinfo, \* fold=0)

>>> from datetime import time

>>> t = time(5, 15, 00)

>>> t

datetime.time(5, 15)

>>> t.replace(21, 29, 45)

datetime.time(21, 29, 45)

এবার আসা যাক টাইম অবজেক্ট ফরম্যাটিংয়ের বিষয়ে। এ জন্য আমরা time.strftime(format) মেথড ব্যবহার করব।

>>> from datetime import time

>>> t = time(5, 15, 00)

>>> t

datetime.time(5, 15)

>>> t.strftime('%I:%M:%S')

'05:15:00'

datetime.datetime()

এই ক্লাসের প্যারামিটার অনেকগুলো- year, month, day, hour=0, minute=0, second=0, microsecond=0, tzinfo=None, \*, fold=0। এর মধ্যে বছর, মাস ও দিন হলো রিকোয়ার্ড প্যারামিটার। tzinfo দিয়ে টাইমজোন ইনফরম্যাশন বোঝায়। এটি নিয়ে এখনই হতাশ হওয়ার কিছু নেই। একটু পরেই আমরা আলোচনা-পর্যালোচনা করব। যাহোক, এই ক্লাসে যথাযথভাবে প্যারামিটার পাস করলে একটা datetime অবজেক্ট পাওয়া যায়, যাতে date অবজেক্ট ও time অবজেক্টের সব ইনফরমেশন থাকে। এবার এই অবজেক্টের ওপর আমরা কয়েকটা মেথড অ্যাপ্লাই করব।

datetime.today() ও datetime.now(tz=None) বর্তমান তারিখ ও সময় (আঞ্চলিক সময়) রিটার্ন করে, মেথড দুটির মধ্যে কোনো পার্থক্য নেই। তবে datetime.now() মেথডে যদি tzinfo সাবক্লাসের কোনো ইনস্ট্যান্স পাস করা হয়, সে ক্ষেত্রে টাইমজোন অনুসারে সবকিছু বদলে যাবে।

>>> from datetime import datetime

>>> datetime.today()

datetime.datetime(2018, 2, 9, 22, 49, 33, 78253)

>>> datetime.now()

datetime.datetime(2018, 2, 9, 22, 49, 49, 886479)

datetime.utcnow() মেথড বর্তমান UTC (গ্রিনউইচ মানমন্দির সাপেক্ষে) তারিখ ও সময় রিটার্ন করবে।

>>> from datetime import datetime

>>> datetime.utcnow()

datetime.datetime(2018, 2, 9, 16, 50, 6, 309991)

ওপরের তিনটি মেথড ব্যবহার করে পাওয়া বা যেকোনো datetime অবজেক্টকে আমরা চাইলে বিভিন্নভাবে অ্যাক্সেস করতে পারি। যেমন:

>>> from datetime import datetime

>>> temp = datetime.now()

>>> temp

datetime.datetime(2018, 2, 9, 22, 50, 25, 678815)

>>> temp.year

2018

>>> temp.month

2

>>> temp.day

9

>>> temp.hour

22

>>> temp.minute

50

>>> temp.second

25

>>> temp.microsecond

678815

datetime.strptime(date\_string, format) মেথডের সাহায্যে একটি ডেট স্ট্রিংকে ফরম্যাট করে datetime অবজেক্টে কনভার্ট করা যায়। আমরা যেহেতু ফরম্যাট কোড শিখেছি। সুতরাং সরাসরি কাজ শুরু করা যাক:

>>> from datetime import datetime

>>> temp = '05:15:00 AM 23 October, 1996'

>>> datetime.strptime(temp, '%I:%M:%S %p %d %B, %Y')

datetime.datetime(1996, 10, 23, 5, 15)

ডেটটাইম অবজেক্টের কোনো অ্যাট্রিবিউটের ভ্যালু চেঞ্জ করতে চাইলে আমাদের কিছু মেথড ব্যবহার করতে হবে। যেমন:

datetime.replace(year=self.year, month=self.month, day=self.day, hour=self.hour, minute=self.minute, second=self.second, microsecond=self.microsecond, tzinfo=self.tzinfo, \* fold=0)

একটি উদাহরণ দেখা যাক :

>>> from datetime import datetime

>>> temp = datetime.now()

>>> temp

datetime.datetime(2018, 2, 9, 22, 51, 58, 304110)

>>> temp.replace(year=1996, month=10, day=23, hour=5, minute=15)

datetime.datetime(1996, 10, 23, 5, 15, 58, 304110)

এবার আসা যাক datetime অবজেক্ট ফরম্যাটিংয়ের বিষয়ে। datetime.strftime(format) মেথডের সাহায্যে আমরা খুব চমৎকারভাবে datetime অবজেক্টকে ফরম্যাট করতে পারি। এখানেও আমাদের ফরম্যাট কোড ব্যবহার করতে হবে।

>>> from datetime import datetime

>>> temp = datetime.now()

>>> temp

datetime.datetime(2018, 2, 9, 22, 51, 58, 304110)

>>> temp.strftime('%I:%M:%S %p %d %B, %Y')

'10:51:58 PM 09 February, 2018'

এবার আমরা একটা জোড়াতালির পদ্ধতি দেখব:

>>> from datetime import datetime, date, time

>>> my\_date = date(1994, 10, 23)

>>> my\_time = time(5, 15, 00)

>>> datetime.combine(my\_date, my\_time)

datetime.datetime(1994, 10, 23, 5, 15)

এখানে আমরা একটি date অবজেক্ট ও একটি time অবজেক্টকে সমন্বিত করে একটি datetime অবজেক্ট তৈরি করেছি।

## time মডিউল

এই মডিউলের সহায়তায় আমরা সময়কে আরও বেশি নিয়ন্ত্রণ করতে পারব। আসলে datetime.time() ক্লাসের চেয়ে time মডিউল আরও বিস্তর এবং ব্যাপক।

### time.gmtime()

time.struct\_time ক্লাসের অবজেক্ট হিসেবে UTC (গ্রিনউইচ মান মন্দির) সময় রিটার্ন করে। যেমন:

>>> import time

>>> time.gmtime()

time.struct\_time(tm\_year=2018, tm\_mon=2, tm\_mday=9, tm\_hour=16, tm\_min=53, tm\_sec=20, tm\_wday=4, tm\_yday=40, tm\_isdst=0)

### time.localtime()

time.struct\_time ক্লাসের অবজেক্ট হিসেবে লোকাল টাইম (বাংলাদেশের ক্ষেত্রে UTC+6) রিটার্ন করে। যেমন:

>>> import time

>>> time.localtime()

time.struct\_time(tm\_year=2018, tm\_mon=2, tm\_mday=9, tm\_hour=22, tm\_min=53, tm\_sec=40, tm\_wday=4, tm\_yday=40, tm\_isdst=0)

### time.strptime(string[, format])

টাইম স্ট্রিংকে ফরম্যাট অনুসারে time.struct\_time ক্লাসের অবজেক্টে কনভার্ট করে। ফরম্যাট কিন্তু আমরা আগেই শিখেছি। ওই এক গানই সব সময় গাইতে হবে। অফিশিয়াল ডক থেকে একটা উদাহরণ দেখা যাক:

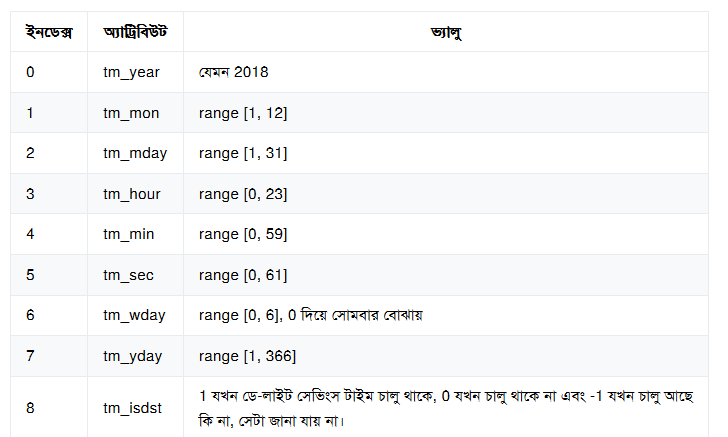
>>> import time

>>> time.strptime("30 Nov 00", "%d %b %y")

time.struct\_time(tm\_year=2000, tm\_mon=11, tm\_mday=30, tm\_hour=0, tm\_min=0, tm\_sec=0, tm\_wday=3, tm\_yday=335, tm\_isdst=-1)

### time.struct\_time

এই ক্লাসের কথা ইতিমধ্যে আলোচনা হয়েছে। gmtime(), localtime() ও strptime() ফাংশন এই ক্লাসের অবজেক্ট রিটার্ন করে যা আসলে একটা টাপল; মূলত TimeTuple হিসেবে সুপরিচিত। অবজেক্টের ভ্যালুগুলো ইনডেক্স বা অ্যাট্রিবিউটের নাম অনুসারে অ্যাক্সেস করা যায়।



একটি উদাহরণ দেখা যাক :

>>> import time

>>> t = time.localtime()

>>> t

time.struct\_time(tm\_year=2018, tm\_mon=2, tm\_mday=9, tm\_hour=22, tm\_min=54, tm\_sec=37, tm\_wday=4, tm\_yday=40, tm\_isdst=0)

>>> t.tm\_year

2018

>>> t[0]

2018

>>> t.tm\_mon

2

>>> t[1]

2

>>> t.tm\_mday

9

>>> t[2]

9

### time.time()

সেকেন্ডের স্কেলে (যেমন 2s, 4.1s) কারেন্ট টাইম রিটার্ন করে (ফ্লোটিং পয়েন্ট নম্বর হিসেবে)। ডেকোরেটর চ্যাপ্টারে কম্পিউটেশনাল কস্ট হিসাব করতে ব্যবহার করেছিলাম আমরা।

import time

start = time.time()

count = 0

while count <= 1000:

count += 1

stop = time.time()

print('Computational cost:', stop-start)

**আউটপুট**

Computational cost: 0.00016188621520996094

### time.clock()

এটি খুব দরকারি একটি ফাংশন। সেকেন্ডের স্কেলে (যেমন 2s, 4.1s) বর্তমান সিপিইউ টাইম রিটার্ন করে (ফ্লোটিং পয়েন্ট নম্বর হিসেবে)। বিভিন্ন কম্পিউটেশনাল কস্ট হিসাব করতে time.time()-এর চেয়ে বেশি কার্যকরী। যেমন:

import time

start = time.clock()

count = 0

while count <= 1000:

count += 1

stop = time.clock()

print('Computational cost:', stop-start)

**আউটপুট**

Computational cost: 0.0001735687255859375

### time.sleep(secs)

ঝিমাইন্না মুরগি। থুরি, ঝিমাইন্না প্রোগ্রাম বানানোর ক্ষেত্রে এই ফাংশন প্রয়োগ করা হয়। কখন‌ো কখনো কোনো প্রোগ্রামের মধ্যে কলিং থ্রেডকে কিছুক্ষণের জন্য ঝিম ধরিয়ে রাখার দরকার পড়ে। অর্থাৎ ওই সময় ওই ব্লকে এসে প্রোগ্রামটি এক্সিকিউট হবে না। এ কাজের জন্য time.sleep() ফাংশন ব্যবহার করা হয়।

import time

start = time.time()

time.sleep(3)

stop = time.time()

print('Computational cost:', stop-start)

**আউটপুট**

Computational cost: 3.003114938735962

আউটপুট দেখেই বোঝা যাচ্ছে যে প্রোগ্রামটা ৩ সেকেন্ড ঝিমাইছে, উমাইন্না মুরগির মতো!

### time.strftime(format[, t])

টাইম টাপলকে স্ট্রিংয়ে ফরম্যাট করার জন্য আমরা এই ফাংশন ব্যবহার করব। ফরম্যাট ওই আগেরগুলোই। একটি উদাহরণ দেখা যাক:

>>> import time

>>> t = time.gmtime()

>>> t

time.struct\_time(tm\_year=2018, tm\_mon=2, tm\_mday=9, tm\_hour=16, tm\_min=57, tm\_sec=21, tm\_wday=4, tm\_yday=40, tm\_isdst=0)

>>> time.strftime('%I:%M:%S %p %d %B, %Y', t)

'04:57:21 PM 09 February, 2018'

time.strptime(string[, format])

এ ফাংশনটি হলো time.strftime() ফাংশনের উল্টা। একটি ডেট-টাইম স্ট্রিংকে টাইমটাপলে কনভার্ট করার জন্য এ ফাংশনটি ব্যবহার করা হয়।

>>> import time

>>> time.strptime('04:57:21 PM 09 February, 2018', '%I:%M:%S %p %d %B, %Y')

time.struct\_time(tm\_year=2018, tm\_mon=2, tm\_mday=9, tm\_hour=16, tm\_min=57, tm\_sec=21, tm\_wday=4, tm\_yday=40, tm\_isdst=-1)

এই হলো মোটামুটি কাহিনি। টাইমজোন জটিলতায় আমরা গেলাম না। আপাতত এই চ্যাপ্টারের অংশটুকু জানলেই আমাদের কাজ চলবে। তবে পাইথনের অফিশিয়াল ডকুমেন্ট থেকে datetime নিয়ে আরও জানতে <https://docs.python.org/3/library/datetime.html> এই লিংকে এবং time নিয়ে আরও জানতে এই লিংকে - <https://docs.python.org/3/library/time.html> ঘুরে আসতে পারেন।

## পাইথনের “ইম্পোর্ট” নিয়ে কিছু কথা

**লাইব্রেরি** হচ্ছে এমন এক প্যাকেজ যা অন্যকোন প্যাকেজ এর অধীনে নয়। অর্থাৎ টপ হায়েরারকিতে থাকে। অনেকগুলি প্যাকেজ, সাব প্যাকেজ আর মডিউল এর সমষ্টিও বলা যেতে পারে লাইব্রেরিকে। সাধারণত জ্যাঙ্গো বা অন্য কোন ফ্রেমওয়ার্ক এর মূল ফোল্ডারটিই লাইব্রেরি হিসেবে গণ্য হয়। জেনে রাখুন, উদাহরনে গেলেই বুঝে যাবেন একটু পরে।

**প্যাকেজ** হচ্ছে এমন একটি ফোল্ডার স্ট্রাইকচার, যার মাঝে ইনিশিয়ালাইজেশন ফাইল “\_\_init\_\_” থাকবে এবং আরো সাব প্যাকেজ অথবা মডিউল থাকতে পারে। এটা হচ্ছে লাইব্রেরির পরবর্তী চাইল্ড লেভেল।

**সাব প্যাকেজ** সাধারণ প্যাকেজই, কিন্তু এগুলি অন্য কোন প্যাকেজ এর আন্ডারে থাকে। পাইথনে অফিসিয়ালি সাব প্যাকেজ বলে কিছু নেই, সাব প্যাকেজ গুলিও প্যাকেজ। শুধু বোঝার সুবিধার জন্য বলা।

**মডিউল** হচ্ছে সিঙ্গেল পাইথন ফাইল, যার মাঝে ক্লাস, মেথড, ফাংশন, ভেরিয়েবল এনকেপসুলেট করা থাকে।

একটা উদাহরণ দেই চলুন।

from django.utils import timezone

এখানে কোনটা কি আসুন একটু ব্রেকডাউন করি।

ও হ্যা, বলতে ভুলে গেছি। পাইথনে আপনি অনেকভাবে ইম্পোর্ট করতে পারেন। তন্মধ্যে সবচেয়ে বেশি ব্যবহৃত হলো import আর from….import স্ট্যাটমেন্ট। এছাড়াও as কিওয়ার্ড এর মাধ্যমে আপনি কোন লং বা কঠিন নাম এর ইম্পোর্টকে নিজের মত করে নাম দিতে পারেন।

যাই হোক,

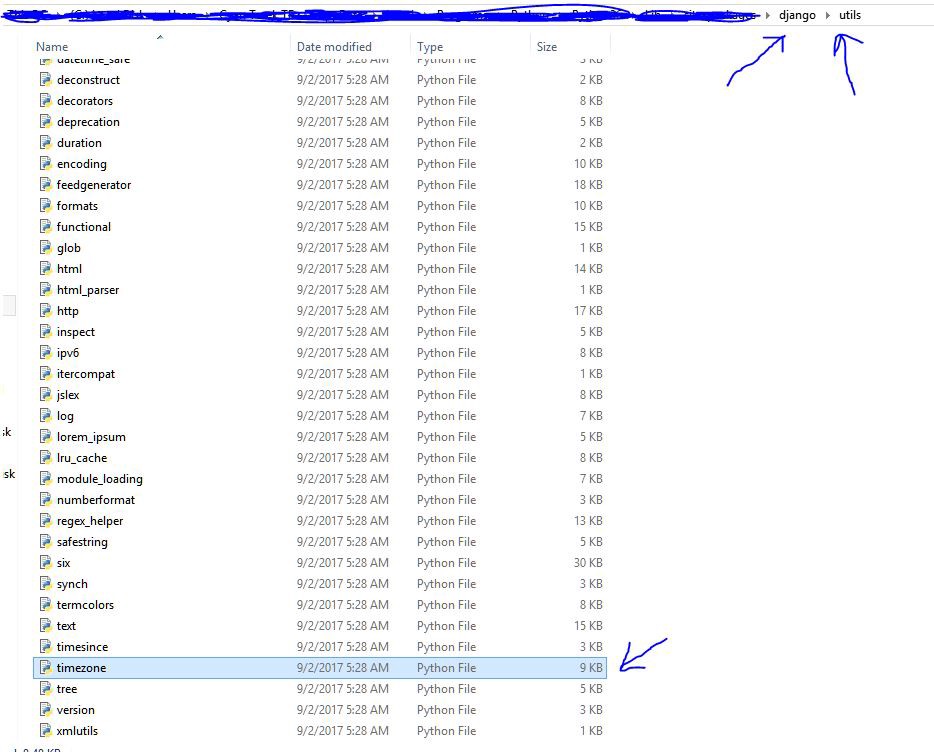
আমরা জানি **django** এখানে হচ্ছে মূল ফ্রেমওয়ার্ক, যা ফোল্ডার স্ট্রাকচারে একদম টপ, অর্থাৎ পেরেন্ট। তাই এটা হচ্ছে মূল লাইব্রেরি।

ডট নোটেশন দিয়ে আমরা ফোল্ডার থেকে -> ফোল্ডার, ফাইল, ক্লাস, মেথড, ফাংশন, ভেরিয়েবলে ওয়াকওভার করতে পারি।

যেমন এখানে আমরা **django** লাইব্রেরি থেকে **utils** প্যাকেজে গেলাম। যদি ঐ প্যাকেজে গিয়েই আপনার কাংখিত জিনিসটি থাকে তাহলে হাটাচলা থামান। অথবা আরো সামনে যেতে থাকুন।

আমরা যেহেতু এখানে কাংখিত ফাইলের কাছে পৌছে গেছি, তাই আমরা এখানে হাটাচলা থামালাম। এরপর আমরা আমাদের কাংখিত **timezone** ফাইলটি ইম্পোর্ট করলাম, এই ফাইলটিকে পাইথনের ভাষায় কি বলা হবে? ঠিক ধরেছেন, মডিউল।

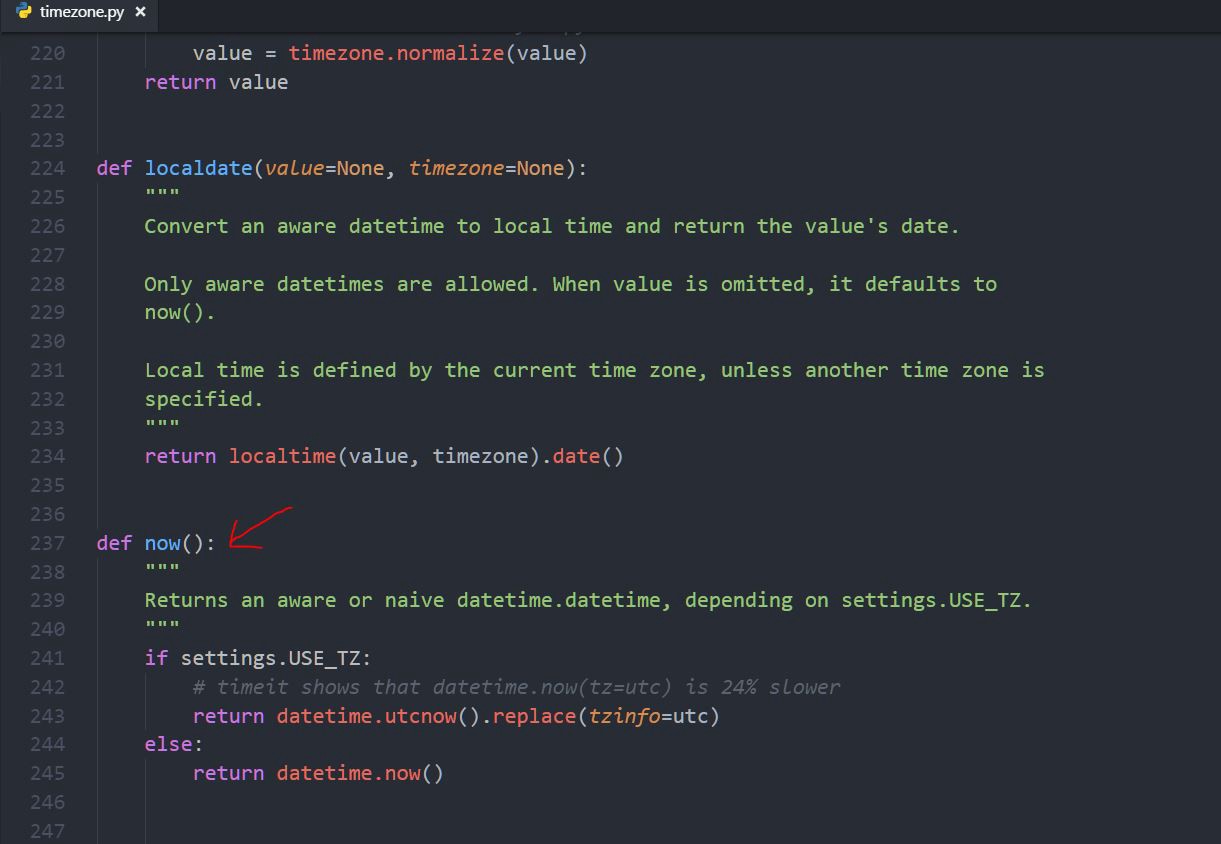
Image for post



এখানে আমরা পুরো মডিউলটিই ইম্পোর্ট করেছি। আমরা চাইলে মডিউলের ভিতরে ওয়াকওভার করে ক্লাস, মেথড ইত্যাদি স্পেফিসিকভাবে ইম্পোর্ট করতে পারতাম।

আরেকটা উদাহরণ দেই। আমরা আমাদের কাজের জন্য এখন কয়টা বাজে তা চেক করার জন্য **now()** ফাংশনটি ব্যবহার করতে পারি। তার আগে আমাদের বের করতে হবে এটা কোথায় আছে। এটা আছে পূর্বের **timezone** মডিউলের ভেতরে। তাহলে স্পেসিফিকভাবে এই ফাংশটা কিভাবে ইম্পোর্ট করবেন?

Image for post



তার আগে বলে নিই এমন স্পেসিফিকভাবে ইম্পোর্ট না করলে কি হয়। তখন আপনি যতবারই **now()** ফাংশনটি ব্যবহার করবো ততবারই আমাদেরকে **timezone.now()** নোটেশন ব্যবহার করতে হবে, কারণ আমরা **timezone** মডিউলটা ইম্পোর্ট করেছি পুরোটা। এটাকে বলে **ইমপ্লিসিট ইম্পোর্ট**, মানে পরিষ্কারভাবে কোন নির্দিষ্ট কিছু ইম্পোর্ট করিনি। যদি আপনার পুরো মডিউলের অনেক কিছু দরকার হয় তাহলে এটা ঠিক আছে। নয়ত, পাইথনের স্ট্যান্ডার্ড রুল অনুযায়ী আপনাকে এক্সপ্লিসিটভাবে ইম্পোর্ট করতে হবেঃ ***Explicit is better than implicit***

যাই হোক, উপরের হাটাচলার নিয়ম মনে রাখলে আর এক্সপ্লিসিট/ইমপ্লিসিট এর কথা জানা থাকলে খুব সহজেই আপনি এবার শুধুমাত্র now() ফাংশনটি ইম্পোর্ট করতে পারেন।

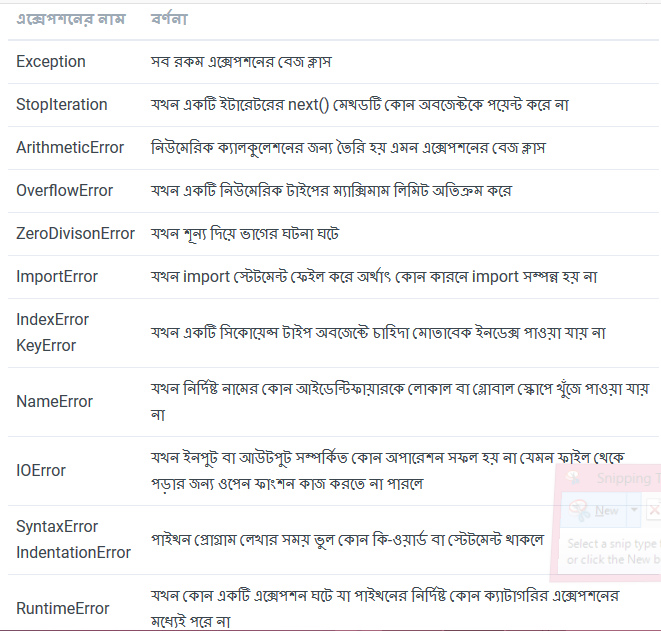
**from django.utils.timezone import now**

খেয়াল করে দেখুন আমরা ফাংশনকে পেরেন্থেসিস দিয়ে now() হিসেবে ইম্পোর্ট করিনি। এই কনসেপ্টকে **ফার্স্ট ক্লাস অবজেক্ট** বলে, অর্থাৎ পাইথনের সবকিছুই অবজেক্ট এবং ফাংশন হলো ফার্স্ট ক্লাস অবজেক্ট, তাই ফাংশনকে আপনি নর্মাল অবজেক্ট এর মতই ব্যবহার করতে পারবেন। অনেকটা সি++ এর পয়েন্টার/রেফারেন্স এর মত। এ নিয়ে আরেকদিন লিখবো। আজকে এই পর্যন্তই। আল্লাহ হাফেজ।

# এক্সেপশন

এটি এমন একটি ইভেন্ট যা ঘটে তখনই, যখন একটি প্রোগ্রামের স্বাভাবিক এক্সিকিউশনের মধ্যে কোন বাধার উৎপত্তি হয়। অর্থাৎ যখন একটি পাইথন স্ক্রিপ্ট এমন কোন একটি সমস্যাপূর্ণ অবস্থার সম্মুখীন হয় যা সে এড়িয়ে যেতে পারে না অথবা সমাধান করতে পারে না অতঃপর প্রোগ্রামের এক্সিকিউশন বন্ধ হয়ে যায় - সেরকম ঘটনাকে এক্সেপশন বলা হয়। এক্সেপশন এর আভিধানিক অর্থ থেকেও বোঝা যায় যে ব্যতিক্রম কোন অবস্থার উৎপত্তি।

সাধারণত ভুল কোড বা ইনপুটের জন্য প্রোগ্রামের মধ্যে এক্সেপশন তৈরি হয় যা সঠিকভাবে হ্যান্ডেল না করলে প্রোগ্রাম অনাকাঙ্ক্ষিত ভাবে বন্ধ হয়ে যেতে পারে। একটি উদাহরণ দিয়ে আমারা বোঝার চেষ্টা করি



## এক্সেপশন হ্যান্ডেলিং

আগের চ্যাপ্টারে আমরা দেখেছি, এক্সেপশন তৈরি হলে প্রোগ্রাম অনাকাঙ্ক্ষিত ভাবে বন্ধ হয়ে যায়। খুশির খবর হচ্ছে এরকম তৈরি হওয়া এক্সেপশন গুলোকে সঠিকভাবে হ্যান্ডেল করতে পারলে প্রোগ্রাম যেমন বন্ধ না হয়ে এগিয়ে চলবে তেমনি প্রোগ্রামের কোথায় কোন সমস্যা আছে সেগুলোকেও সহজে চিহ্নিত করা যাবে। এ জন্য পাইথনে আছে try, except স্টেটমেন্টের ব্যবহার।

try ব্লকের মধ্যে এমন কোড গুলো লেখা হয় যেখানে এক্সেপশন তৈরি হতে পারে (ইউজার ইনপুট বা সেরকম অন্যান্য কারনে)। আর except ব্লকের মধ্যে লেখা হয় এমন কোড যেগুলো এক্সিকিউট হবে যদি আসলেই ওই try ব্লকের মধ্যে কোন এক্সেপশন তৈরি হয়। অর্থাৎ try এর মধ্যে এক্সেপশন তৈরি হলে এই ব্লকের কোড এক্সিকিউশন বন্ধ হবে কিন্তু except ব্লকের কোড স্বাভাবিক ভাবে এক্সিকিউট হবে। একটি উদাহরণ দেখি -

**def** fun(**x**,**y**):

**try**:

        print(x**/**y)

**except** ZeroDivisionError:

        print('dont enter zero')

fun(2,2)

fun(2,0) #this argument show zerodivision error

আর তাই সেটুকু আন্দাজ করেই আমরা ভাগ করার কোড টুকু একটি ট্রাই ব্লকের মধ্যে লিখেছি এবং সেই ব্লকের মধ্যে যদি শূন্য দিয়ে ভাগ করার কারনে কোন এক্সেপশন তৈরি হয় তাহলে সেটা হ্যান্ডেল করার জন্য এক্সেপ্ট ব্লক ব্যবহার করেছি এবং নির্দিষ্ট করে ZeroDivisionError এক্সেপশন হ্যান্ডেল করেছি। এখন, ইউজার চাইলে শূন্য ইনপুট দিতে পারে, তাই বলে প্রোগ্রাম অনাকাঙ্ক্ষিত ভাবে শাটডাউন বা বন্ধ হবে না। বরং ইউজারকে যথাযথ ম্যাসেজ দেখিয়ে স্বাভাবিক কাজ চালিয়ে যেতে পারছে।

একটি try ব্লকের সাপেক্ষে একাধিক except ব্লক থাকতে পারে। আবার একটি except এর জন্য একাধিক এক্সেপশন ডিফাইন করা যেতে পারে ব্র্যাকেট এবং কমা ব্যবহার করে। এতে করে ট্রাই ব্লকের মধ্যে বিভিন্ন রকম এক্সেপশনের জন্য বিভিন্ন এক্সেপ্ট ব্লক দিয়ে সঠিক ভাবে সমস্যাকে চিহ্নিত করা যায় এবং সে অনুযায়ী কাজ করা যায়। আরেকটি উদাহরণ দেখি -

*#TpeError*

**def** fun1():

    print(22**+**'hello')

fun1()

*#zerror Division errro*

**def** fun2(**x**,**y**):

    print(x**/**y)

fun2(2,0)

*#NameError*

**def** fun3():

    print(x**/**y)

fun3(2,abc)

example:

**def** fun1(**a**,**b**):

**try**:

        print(a**/**b)

**except** ZeroDivisionError:

        print('dont enter 0 value')

**except** ValueError:

        print('dont enter string value')

**except** TypeError:

        print('dont enter charecter value')

**except** NameError:

        print('dont enter value')

fun1(2,0)

fun1(2,'abc')

fun1(2,a)

example

*#multiple error in one exception*

**def** fun(**x**,**y**):

**try**:

        print(x**/**y)

**except** (ZeroDivisionError,TypeError,NameError,ValueError):

        print('dont enter zero')

        print('dont enter string value')

fun(2,o)

fun(2,a)

fun(2,'a')

## finally

যদি এমন দরকার হয় যে, যতই এক্সেপশন তৈরি হোক না কেন কিছু কোডকে রান করানো দরকার, তখন finally স্টেটমেন্ট ব্যবহার করা হয়। try, except ব্লকের নিচে finally ব্লক ব্যবহার করতে হয়। try বা except ব্লকের কোড রান হবার পর এই finally ব্লকের মধ্যে থাকা কোড গুলো রান হবেই। একটি উদাহরণ দেখি –

**def** fun1(**a**,**b**):

**try**:

        print(a**/**b)

**except** ZeroDivisionError:

        print('dont enter 0 value')

**finally**:

        print('i am must runn')

fun1(2,0)

উপরের প্রোগ্রামে, try ব্লকের মধ্যে প্রথম প্রিন্ট স্টেটমেন্টের পর দ্বিতীয় প্রিন্ট স্টেটমেন্টে শূন্য দিয়ে ভাগের চেষ্টার কারনে ZeroDivisionError এক্সেপশন তৈরি হচ্ছে। সেটাকে সঠিকভাবে হ্যান্ডেল করায় except ব্লকের মধ্যে থাকা print("Divided by zero") এক্সিকিউট করছে। এবং পরিশেষে, যেহেতু ঘটনা যাই হোক finally ব্লক এর কোড এক্সিকিউট হবেই, তাই print("This code will run no matter what") স্টেটমেন্টটিও কাজ করছে।

যদি finally ব্লকের আগে এমন কোন এক্সেপশন তৈরি হয় যাকে সঠিক ভাবে হ্যান্ডেল করা হয় নাই, সে অবস্থাতেও finally ব্লকের কোড রান হবে। যেমন -

**def** fun1(**a**,**b**):

**try**:

        print(a**/**b)

**except** ZeroDivisionError:

        print('dont enter 0 value')

**finally**:

        print('i am must runn')

*#fun1(2,0)*

fun1(2,'ham')

উপরের প্রোগ্রামের try ব্লকের মধ্যে একটি এক্সেপশন তৈরি হয় এবং সেটা except ব্লকে হ্যান্ডেল করা হয়। কিন্তু সেই হ্যান্ডেল করার ব্লকের মধ্যে আবার এমন একটা ভ্যারিয়েবল প্রিন্ট করতে চাওয়া হয়েছে যাকে ডিফাইন করাই হয় নাই। আর তাতে করে সেখানে একটা NameError টাইপের এক্সেপশন তৈরি হয় (যদিও এটাকে হ্যান্ডেল করা হয় নি)। তারপরেও finally ব্লক কাজ করছে আর তাই This is executed last কে আউটপুট স্ক্রিনে দেখা যাচ্ছে।

## মডিউল ও প্যাকেজ

প্রোগ্রাম লেখার সময় আমাদের অনেক রকম কাজ করতে হতে পারে। কী কী কাজ করতে হবে, তা নির্ভর করে আমরা আসলে প্রোগ্রাম দিয়ে কী করতে চাচ্ছি তার উপরে। যেমন, আমরা গাণিতিক সমস্যার সমাধানের জন্য প্রোগ্রাম লিখতে পারি, অপারেটিং সিস্টেমের সঙ্গে কাজ করার জন্য প্রোগ্রাম লিখতে পারি, ইন্টারনেটে যুক্ত হয়ে ডেটা আদান-প্রদানের জন্য প্রোগ্রাম লিখতে পারি, ছবি-ভিডিও ইত্যাদি নিয়ে কাজ করতে পারি, কোনো গেমস তৈরি করতে পারি। এরকম কত কী যে করা যায় প্রোগ্রামিং করে, তার হিসেব নেই। এখন প্রোগ্রামারদের কাজ সহজ করার জন্য অনেক মডিউল তৈরি করে দেওয়া আছে। আবার একাধিক মডিউল একসঙ্গে করে প্যাকেজও তৈরি করে দেওয়া আছে। মডিউলগুলোতে থাকে বিভিন্ন ডেটা, ফাংশন ও ক্লাস (class)। ক্লাস কী জিনিস, সেটি আমরা এই বইতে পরে শিখবো।

আমরা যে বিভিন্ন প্রোগ্রামে print(), input(), type(), sum() ইত্যাদি ফাংশন ব্যবহার করেছি, সেগুলো কিন্তু আমরা নিজেরা তৈরি করি নি, ব্যবহার করেছি কেবল। এগুলোকে বিল্টইন (builtin) ফাংশন বলে। বিল্টইন শব্দের অর্থ হচ্ছে, যেগুলো কোনোকিছুর সঙ্গে দেওয়া থাকে। যেমন, টেলিভিশন অন-অফ করার জন্য বিল্টইন সুইচ দেওয়া থাকে। ওই ফাংশনগুলো তৈরি করা না থাকলে আমাদের নিজেদের সেগুলো তৈরি করা লাগত এবং প্রতিবার আমাদের বেশ খানিকটা সময় ব্যায় হতো। ফাংশনগুলো ইতিমধ্যে তৈরি করা আছে builtins নামক মডিউলে। পাইথন ইন্টারপ্রেটার চালু করলে এই মডিউল আপনাআপনি চলে আসে, নইলে আমাদেরকে আলাদাভাবে ইমপোর্ট (import) করতে হতো। কারো যদি জানতে ইচ্ছে করে যে, এই মডিউলে কী কী আছে, তাহলে সেটি জানার জন্য পাইথন ইন্টারপ্রেটার চালু করে নিচের দুই লাইন লিখতে হবে :

>>> import builtins

>>> dir(builtins)

dir() নিজেও একটি বিল্টইন ফাংশন, যার ভেতরে কোনো মডিউলের নাম আর্গুমেন্ট হিসেবে পাঠালে ওই মডিউলে কী কী গ্লোবাল ভ্যারিয়েবল, ফাংশন, ক্লাস ইত্যাদি আছে এগুলো সে একটি লিস্টে রিটার্ন করে।

#### স্ট্যান্ডার্ড লাইব্রেরি (Standard Library)

বিল্টইন ফাংশন ছাড়াও কিন্তু আমরা বিভিন্ন মডিউল ইমপোর্ট করে ব্যবহার করেছি। যেমন, math মডিউল ইমপোর্ট করেছি মৌলিক সংখ্যা বের করার প্রোগ্রাম লেখার সময়। এজন্য আমাদের লিখতে হয়েছে, import math। এই মডিউলটি হচ্ছে পাইথনের স্ট্যান্ডার্ড লাইব্রেরির একটি মডিউল। এখানে বিভিন্ন গাণিতিক ফাংশন তৈরি করে দেওয়া আছে, সেই সঙ্গে কিছু গাণিতিক ধ্রুবক (constant)-ও সংজ্ঞায়িত করে দেওয়া আছে। কিছু উদাহরণ দেখি :

>>> import math

>>> math.pi

3.141592653589793

>>> math.pow(2, 3)

8.0

>>> math.pow(3, 2)

9.0

>>> math.sqrt(25)

5.0

>>> math.floor(5.2)

5

>>> math.ceil(5.2)

6

math মডিউলের কী কী ফাংশন আছে এবং সেগুলোর কাজ কী, তা পুরোপুরিভাবে জানতে হলে ডকুমেন্টেশন দেখতে হবে। পাইথনের ওয়েবসাইটে ডকুমেন্টেশন দেওয়া আছে ([https://docs.python.org/3/library/math.html\](https://docs.python.org/3/library/math.html/)) আবার চাইলে সেটি ডাউনলোডও করে ফেলা যায়। math ছাড়াও আমরা ইতিপূর্বে পাইথন দিয়ে প্রোগ্রামিং শেখা বইতে turtle, random, timeit ইত্যাদি মডিউল ব্যবহার করেছি। আরো কয়েকটি উদাহরণ দেখাই।

দিন-তারিখ নিয়ে কাজ-কর্ম করার জন্য আছে datetime মডিউল। এই মডিউলের মধ্যে আবার বিভিন্ন ক্লাস আছে। আমরা সেই ক্লাসের বিভিন্ন মেথড ব্যবহার করতে পারি। এখন আমরা যদি আজকের তারিখ জানতে চাই, তাহলে datetime মডিউলের date ক্লাসের today() মেথডকে কল করতে হবে।

>>> import datetime

>>> today = datetime.date.today()

>>> print(today)

2017-06-10

আবার আমরা যদি আজকের তারিখ ও বর্তমান সময় জানতে চাই, তখন datetime মডিউলের datetime ক্লাসের today() মেথড কল করতে হবে।

>>> import datetime

>>> today = datetime.datetime.today()

>>> print(today)

2017-06-10 15:51:28.951772

ওপরের দুটি উদাহরণে আমরা দেখলাম প্রথমে মডিউলের নাম, তারপরে একটি ডট, তারপরে ক্লাসের নাম, তারপরে একটি ডট ও সবশেষে ফাংশনের নাম লিখে আমরা ফাংশনটি কল করছি। আবার মডিউলটি যদি কোনো প্যাকেজের মধ্যে থাকত, তাহলে প্রথমে সেই প্যাকেজের নাম লিখে ডট দিতে হতো এবং তারপরে মডিউলের নাম, ক্লাসের নাম, ফাংশনের নাম ইত্যাদি লিখতে হতো। অর্থাৎ, package\_name.module\_name.function\_name() আর যদি মডিউলে কোনো ক্লাসের মেথডকে কল করি, তাহলে package\_name.module\_name.class\_name.function\_name()।

আগের উদাহরণে আমরা চাইলে datetime মডিউল থেকে কেবল datetime ক্লাসটি ইমপোর্ট করতে পারতাম। তখন datetime.today() লিখলেই কাজ হয়ে যাবে।

>>> from datetime import datetime

>>> d = datetime.today()

>>> print(d)

2017-06-10 17:08:56.428959

আমরা যদি আমাদের প্রোগ্রাম থেকে কোনো ওয়েবসাইট, ওয়েব ব্রাউজারের মাধ্যমে দেখাতে চাই, তার জন্যও ফাংশন তৈরি করে দেওয়া আছে webbrowser মডিউলে।

>>> import webbrowser

>>> url = "http://subeen.com"

>>> webbrowser.open(url)

True

webbrowser মডিউলের open() ফাংশনটি ব্যবহার করে আমরা ওয়েবসাইট ওপেন করতে পারি। এখন webbrowser শব্দটি পুরোটা বার বার টাইপ করা অনেকের কাছে বিরক্তিকর লাগতে পারে। তার জন্য একটি শর্টকাটও আছে, যাকে বলে এলিয়াসিং (aliasing)। নিচের উদাহরণ দেখলেই বুঝতে পারবো :

>>> import webbrowser as wb

>>> url = "http://subeen.com"

>>> wb.open(url)

True

পাইথনের স্ট্যান্ডার্ড লাইব্রেরিতে কী কী মডিউল ও প্যাকেজ আছে, তা জানা যাবে এখানে : [https://docs.python.org/3/library/index.html।](https://docs.python.org/3/library/index.html%E0%A5%A4) অবসর সময়ে এগুলো নিয়ে পড়াশোনা করা যেতে পারে, তবে তাড়াহুড়ো নেই। আর সব লাইব্রেরির ব্যবহারও শেখার বা মুখস্থ করার দরকার নেই। সময়ের সঙ্গে প্রয়োজনমতো আমরা এমনিতেই শিখে নেবো।

আমাদের মাঝে-মধ্যে এমন কাজ করার দরকার পরবে, যেগুলোর জন্য কেউ একজন মডিউল বা প্যাকেজ তৈরি করে রেখেছে, কিন্তু সেগুলো পাইথনের স্ট্যান্ডার্ড লাইব্রেরি অন্তর্ভুক্ত নয়। এগুলো বলে থার্ড পার্টি প্যাকেজ বা মডিউল। এগুলো ব্যবহার করতে হলে কম্পিউটারে আলাদাভাবে ইনস্টল করতে হয়। আমাদের দরকার মতো আমরা সেগুলো ইনস্টল করে নিবো। এই বইতে পরবর্তি অংশে আমরা দেখবো, কিভাবে থার্ড পার্টি মডিউল ইনস্টল ও ব্যবহার করতে হয়।

#### নতুন মডিউল তৈরি করা

পাইথন দিয়ে প্রোগ্রামিং শেখা বইয়ের শেষ উদাহরণে আমরা দেখেছিলাম, কিভাবে ফিবোনাচ্চি সংখ্যা বের করতে হয়। n-তম ফিবোনাচ্চি সংখ্যা বের করার জন্য আমরা এখন একটি ফাংশন লিখে ফেলি।

def find\_fib(n):

if n <= 2:

return 1

fib\_x, fib\_next = 1, 1

i = 3

while i <= n:

i += 1

fib\_x, fib\_next = fib\_next, fib\_x + fib\_next

return fib\_next

for x in range(1, 11):

print(find\_fib(x))

আমরা find\_fib() নামে একটি ফাংশন তৈরি করলাম, যেটির প্যারামিটার হচ্ছে n এবং ফাংশনটি n-তম ফিবোনাচ্চি সংখ্যা রিটার্ন করে। তারপর একটি লুপ চালিয়ে আমরা ওই ফাংশন কল করে প্রথম দশটি ফিবোনাচ্চি সংখ্যা প্রিন্ট করলাম। আউটপুট আসবে নিচের মতো – প্রথম দশটি ফিবোনাচ্চি সংখ্যা :

$ python fibo.py

1

1

2

3

5

8

13

21

34

55

অনুশীলনী :

* ওপরের প্রোগ্রামে return fib\_next না লিখে return fib\_x লিখলে কী সমস্যা হতো? এটি নিয়ে একটু চিন্তাভাবনা করতে হবে।
* একটি ফাংশন লিখতে হবে, যার কাজ হচ্ছে প্যারামিটার হিসেবে n নেওয়া এবং প্রথম n-সংখ্যক ফিবোনাচ্চি সংখ্যার লিস্ট রিটার্ন করা। ফাংশনটির নাম যদি হয় list\_fib(), তাহলে print(list\_fib(10)) লিখলে আউটপুট আসবে :

[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]

list\_fib() ফাংশনটিও আমরা fibo.py ফাইলে লিখবো। অধিকাংশ পাঠকেরই ফাংশনটি লিখতে ঘণ্টাখানেক চেষ্টা করতে হতে পারে, তাতে হতাশ হওয়ার কিছু নেই। প্রোগ্রামিং শেখার সময় যে যত বেশি চেষ্টা করবে, চিন্তাভাবনা করবে, কোড লিখবে, তার প্রোগ্রামিং শেখা তত ভালো হবে। এক-দুই ঘণ্টা চেষ্টা করেও list\_fib() ফাংশনটি লেখতে না পারলে সামনে আগাতে হবে, আমি ফাংশনটি একটু পরেই লিখে দিয়েছি।

তাহলে আমার fibo.py ফাইলটি দাঁড়াচ্ছে এরকম :

def find\_fib(n):

if n <= 2:

return 1

fib\_x, fib\_next = 1, 1

i = 3

while i <= n:

i += 1

fib\_x, fib\_next = fib\_next, fib\_x + fib\_next

return fib\_next

def list\_fib(n):

fib\_list = [1, 1]

if n <= 2:

return fib\_list[:n]

fib\_x, fib\_next = 1, 1

i = 3

while i <= n:

i += 1

fib\_x, fib\_next = fib\_next, fib\_x + fib\_next

fib\_list.append(fib\_next)

return fib\_list

for x in range(1, 11):

print(find\_fib(x))

print(list\_fib(1))

print(list\_fib(2))

print(list\_fib(10))

এবারে প্রথম কাজ হচ্ছে প্রোগ্রামটি নিজ নিজ কম্পিউটারে টাইপ করে রান করে দেখা। আউটপুট আসবে এরকম :

$ python fibo.py

1

1

2

3

5

8

13

21

34

55

[1]

[1, 1]

[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]

আমি ফাইলের শেষে প্রিন্টগুলো করে নিশ্চিত হওয়ার চেষ্টা করেছি যে, ফাংশনদুটি ঠিকঠাক কাজ করে। এখন আমরা fibo.py ফাইলটি যে ডিরেক্টরি বা ফোল্ডারে আছে, সেখানে আরেকটি ফাইল তৈরি করবো, program.py আর সেখানে নিচের মতো কোড লিখবো :

import fibo

print("Hello, I am inside program.py!")

n = fibo.find\_fib(15)

print("15th fibonacci number is,", n)

এই প্রোগ্রামে আমি আসলে কী করেছি? প্রথমে fibo.py ফাইলটি ইমপোর্ট করেছি। ইমপোর্ট করার সময় ফাইলের এক্সটেনশন .py দেওয়ার প্রয়োজন নেই, তাই আমরা লিখেছি import fibo। এটি করার উদ্দেশ্য হচ্ছে fibo.py ফাইলের ভেতরে যেই ফাংশন দুটি তৈরি করেছি, সেগুলো যেন আমি ব্যবহার করতে পারি। fibo.py ফাইলে দুটি ফাংশন আছে – find\_fib() ও list\_fib()। ফাংশনদুটিকে কল করতে হলে আমাদের লিখতে হবে, যথাক্রমে fibo.find\_fib() ও fibo.list\_fib()। প্রোগ্রামের তৃতীয় লাইনে লিখেছি n = fibo.find\_fib(15), অর্থাৎ আমি 15-তম ফিবোনাচ্চি সংখ্যা বের করে, সেই সংখ্যাটি n-এর মধ্যে রাখছি। তারপরের লাইনে সেটি প্রিন্ট করছি। এখন আমরা প্রোগ্রামটি রান করে দেখি, আউটপুট কী আসে?

$ python program.py

1

1

2

3

5

8

13

21

34

55

[1]

[1, 1]

[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]

Hello, I am inside program.py!

15th fibonacci number is, 610

আমাদের প্রোগ্রামে দুটি লাইন প্রিন্ট করার কথা, সেগুলো ঠিকই প্রিন্ট হয়েছে। কিন্তু তার আগে আবার fibo.py প্রোগ্রামে যা যা প্রিন্ট করতে দিয়েছিলাম, সেগুলোও প্রিন্ট হয়েছে। কেন এমন হলো? একটু তদন্ত করে দেখা যাক। আমরা এখন আমাদের program.py ফাইলে, শেষ দুটি স্টেটমেন্ট বাদ দিয়ে দেবো। ফাইলটি হবে এরকম :

import fibo

print("Hello, I am inside program.py!")

এখন প্রোগ্রাম রান করলে আউটপুট আসবে এরকম :

$ python program.py

1

1

2

3

5

8

13

21

34

55

[1]

[1, 1]

[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]

Hello, I am inside program.py!

তাহলে আমরা বুঝতে পারছি, যখনই আমরা import fibo লিখছি, পাইথন করছে কী, fibo.py ফাইলে রান করছে, তাই ওই ফাইলের শেষে আমি যেই প্রিন্ট স্টেটমেন্টগুলো লিখেছিলাম, সেগুলোর আউটপুটও আমরা দেখতে পাচ্ছি।

এখানে fibo.py ফাইলটিকে বলা হয় মডিউল (module)। একটি পাইথন মডিউলে সাধারণত বিভিন্ন ভেরিয়েবল ও ফাংশন ডেফিনেশন থাকে। মডিউলের ভেতরে ক্লাসও থাকতে পারে (ক্লাস সম্পর্কে আমরা এই বইতে পরে জানবো)। মডিউলটি ইমপোর্ট করে তার ভেতরের জিনিসগুলো ব্যবহার করা যায়। একটু আগেই আমরা তা দেখেছি। কিন্তু একটি বিষয় আমাদের ভাবিয়ে তুলছে। আমরা চাই যে, fibo.py ফাইলটি যদি সরাসরি রান করি, তাহলে যেন তার ভেতরে ফাংশনগুলো কল করে যে প্রিন্টগুলো করেছিলাম, সেগুলো চলবে, কিন্তু অন্য ফাইল থেকে fibo-কে ইমপোর্ট করলে ওই স্টেটমেন্টগুলো চলবে না। এজন্য আমাদেরকে একটি গ্লোবাল (global) ভ্যারিয়েবলের সাহায্য নিতে হবে, যার নাম হচ্ছে, \_\_name\_\_। আর এই গ্লোবাল ভ্যারিয়েবলটি কিন্তু আমরা নিজেরা ডিক্লেয়ার করবো না, প্রোগ্রাম রান করার সময় পাইথন নিজে থেকেই এর মান ঠিক করে দেয়। এখন আমরা trial.py নামে একটি ফাইল তৈরি করে এক লাইন কোড লিখবো। fibo.py ও program.py ফাইলদুটি যে ডিরেক্টরিতে আছে, trial.py ফাইলটিও সেই ডিরেক্টরিতে রাখতে হবে।

print("My name is", \_\_name\_\_)

এখন প্রোগ্রামটি রান করি :

$ python trial.py

My name is \_\_main\_\_

মডিউলের নাম তো হওয়ার কথা ছিল trial, কিন্তু আউটপুটে দেখছি \_\_main\_\_। এর কারণ হচ্ছে আমি যখন trial.py ফাইলটি সরাসরি রান করছি, তখন পাইথন তাকে কোনো মডিউল হিসেবে বিবেচনা করছে না, তাই তার নাম দেখাচ্ছে \_\_main\_\_। এখন আমি program.py ফাইলে trial মডিউল ইমপোর্ট করবো।

import trial

print("Hello, I am inside program.py!")

print(trial.\_\_name\_\_)

প্রোগ্রামটি রান করলে আউটপুট আসবে এরকম :

$ python program.py

My name is trial

Hello, I am inside program.py!

trial

প্রথম লাইনে যখন import trial লিখেছি, তখন trial.py ফাইলের প্রিন্ট স্টেটমেন্ট রান করে আউটপুট দিয়েছে My name is trial। এখন কিন্তু সে তার মডিউলের নাম ঠিকঠাক প্রিন্ট করলো। কারণ অন্য কেউ তাকে ইমপোর্ট করেছে। তাহলে আমরা দেখতে পাচ্ছি যে বাইরে থেকে ইমপোর্ট করলে সঠিক নাম দেখায়, আর ফাইলটি সরাসরি চালালে নাম দেখায় \_\_main\_\_। এই ব্যাপারটি আমরা কাজে লাগিয়ে এমন ব্যবস্থা করতে পারি যেন trial.py সরাসরি রান করলে Hello from trial কথাটি প্রিন্ট হবে, কিন্তু বাইরে থেকে ইমপোর্ট করলে এটি প্রিন্ট হবে না। trial.py ফাইলের কোড হবে এরকম :

if \_\_name\_\_ == "\_\_main\_\_":

print("Hello from trial")

প্রোগ্রামটি রান করি :

$ python trial.py

Hello from trial

এবারে program.py ফাইলটি রান করি :

$ python program.py

Hello, I am inside program.py!

trial

এখন কিন্তু আর trial.py ফাইলের print স্টেটমেন্টটি রান করছে না, কারণ if কন্ডিশনের ভেতরে \_\_name\_\_ -এর মান হচ্ছে trial, তাই শর্তটি মিথ্যা হয়ে যাচ্ছে।

**অনুশীলনীঃ**fibo.py ফাইলটি এমনভাবে পরিবর্তন করতে হবে, যেন ফাইলটি মডিউল আকারে ইমপোর্ট করলে ফাইলের শেষের প্রিন্টগুলো না দেখায়, কিন্তু fibo.py ফাইলটি সরাসরি রান করলে যেন প্রিন্টগুলো দেখায়।

আমরা এতক্ষণ সবগুলো ফাইল একই ডিরেক্টরিতে রেখে কাজ করেছি। তার কারণ কী? কোনো প্রোগ্রামে যখন কোনো মডিউল ইমপোর্ট করা হয়, তখন পাইথন কিভাবে জানবে যে, মডিউলের কোড কোন জায়গায় আছে? যখন কোনো মডিউল ইমপোর্ট করা হয় তখন পাইথন প্রথমে খুঁজে দেখে ওই নামে কোনো বিল্ট-ইন মডিউল আছে কি না। ওই নামে কোনো বিল্ট-ইন মডিউল না থাকলে তার সার্চ পাথের সবগুলো ডিরেক্টরি ও তাদের সাব-ডিরেক্টরির মধ্যে একে একে খুঁজতে থাকে যে, সেই মডিউলটি পাওয়া যায় কি না। সার্চ পাথ (search path) হচ্ছে মূলত অনেকগুলো ডিরেক্টরির একটি তালিকা বা লিস্ট। সার্চ পাথের তালিকার প্রথম এলিমেন্ট বা উপাদান হচ্ছে ”, যেটা বোঝায় বর্তমান ডিরেক্টরি (কারেন্ট ওয়ার্কিং ডিরেক্টরি – **c**urrent **w**orking **d**irectory বা cwd), যেখানে আমরা আমাদের প্রোগ্রামটি চালাচ্ছি। সার্চ পাথের সবগুলো ডিরেক্টরির লিস্ট দেখতে চাইলে আমাদেরকে নিচের কোড ব্যবহার করতে হবে।

>>> import sys

>>> sys.path

>>> # এখানে একটি ডিরেক্টরির লিস্ট আসবে

আমরা চাইলে আমাদের মডিউলটি সার্চ পাথের যেকোনো একটি ডিরেক্টরিতে রেখে দিতে পারি, তাহলে পরবর্তি সময়ে পাইথন সেটিকে সহজে খুঁজে পাবে। আবার যদি এমন হয় যে, আমি একটি মডিউল বা প্যাকেজ তৈরি করলাম, যা আরো অনেক প্রোগ্রামারের কাজে লাগতে পারে, তখন আমি সেই মডিউল বা প্যাকেজটি পাবলিশ করতে পারি। তবে এই বইতে আমি সেটি দেখাবো না। কারো প্রয়োজন হলে সে শিখে নেবে, আমি বিষয়টি জানিয়ে রাখলাম মাত্র।

নোট: একটি মডিউল যদি বেশ বড়সড় হয়ে যায়, তখন তাকে আবার বিভিন্ন আলাদা ফাইলে ভেঙ্গে একটি ডিরেক্টরির মধ্যে রেখে একটি প্যাকেজ তৈরী করা যায়। একটি মডিউলে যেমন, একাধিক ভ্যারিয়েবল, ফাংশন, ক্লাস ইত্যাদি থাকতে পারে; তেমনি, একটি প্যাকেজের মধ্যে একাধিক ফাইল বা মডিউল, সাব-ডিরেক্টরি (যাদেরকে সাব-প্যাকেজও বলা হয়) থাকতে পারে। প্যাকেজের মূল ডিরেক্টরির মধ্যে \_\_init\_\_.py নামে একটি ফাইল তৈরী করে তার মধ্যে বলে দিতে হয়, এই প্যাকেজের মধ্যে কী কী সাব-প্যাকেজ ও মডিউল রয়েছে।

## এক্সেপশন Raise

আগে আমরা দেখেছি কিভাবে পাইথন প্রয়োজনে নিজে থেকেই প্রোগ্রামে কিছু এক্সেপশন তৈরি করে। চাইলে ম্যানুয়ালি কোড লিখেও প্রোগ্রামের নির্দিষ্ট কোন যায়গায় এক্সেপশন রেইজ বা সহজ ভাবে বলতে গেলে এক্সেপশন তৈরি করা যায়। raise স্টেটমেন্ট ব্যবহার করে এভাবে কাস্টম এক্সেপশন তৈরি করা যায়। নিচের উদাহরণটি দেখি –

**def** fun():

**raise** TypeError('HiThere')

**raise** ZeroDivisionError('HiThere')

**raise** NameError('HiThere')

**raise** EOFError('HiThere')

fun()

print("Hello")

raise NameError('HiThere')

উপরের প্রোগ্রামের দ্বিতীয় লাইনে আমরা ম্যানুয়ালি একটি NameError টাইপের এক্সেপশন তৈরি করেছি যার কারনে পাইথন সাধারণভাবেই সেই এক্সেপশনটি থ্রো করেছে।

উপরের মত এক্সেপশনের আর্গুমেন্ট (HiThere) সেট না করেও শুধু NameError এক্সপশন থ্রো করে যেত। যেমন নিচের মত -

raise TypeError

raise এর একটি মজার ব্যবহার দেখবো নিচের উদাহরণে,

**try**:

    num **=** 5 **/** 0

**except**:

    print("Custom message about an error!")

**raise**

খেয়াল করুন কি ঘটছে উপরের প্রোগ্রামে। খুব সহজেই বোঝা যাচ্ছে যে try ব্লকে একটি এক্সেপশন ঘটছে। এটাও বুঝতে পারছি যে সেটা ZeroDivisionError এক্সেপশন হতে পারে কারন শূন্য দিয়ে ৫ কে ভাগ করার চেষ্টা করা হয়েছে। কিন্তু আমরা except ব্লকে নির্দিষ্ট করে কোন এক্সেপশন ডিফাইন করে সেটা হ্যান্ডেল করছি না। তারপরেও শেষ নাগাদ পাইথন আমাদেরকে ZeroDivisionError: integer division or modulo by zero এক্সেপশন দেখাতে পারছে। এর কারন - আমরা except এর মধ্যে raise ব্যবহার করেছি। এভাবেও raise কে কাজে লাগিয়ে এর আগে ঘটে যাওয়া এক্সেপশনের টাইপ পেয়ে যেতে পারি।

## রেইজ এক্সেপশন

পাইথনে তো অনেক বিল্ট-ইন এক্সেপশন আছে ( <https://docs.python.org/3/library/exceptions.html> )। আমরা চাইলে এদেরকে নিজ থেকে রেইজ করতে পারি।

try:

raise NameError('Hey! It is a custom error message.')

except NameError as e:

print(e)

**আউটপুট**

Hey! It is a custom error message.

এক্সেপশন রেইজ করার জন্য raise স্টেটমেন্ট ব্যবহার করতে হয়। raise এর পরে বিল্ট-ইন এক্সেপশনের নাম দিয়ে এর সাথেই ব্রাকেটের ভিতর এরর মেসেজ স্ট্রিং হিসাবে পাস করতে হয়।

বিল্ট-ইন এক্সেপশন ছাড়াও পাইথনে ইউজার-ডিফাইন্ড এক্সেশন ব্যবহার করা যায়। কিন্তু আমরা তো এখনও ছোট। তাই এখনই আমরা এটা শিখব না। ক্লাস, অবজেক্ট, মেথড সম্পর্কে যখন আমাদের ধারণা পরিষ্কার হবে তখন আমরা ইউজার-ডিফাইন্ড এক্সেপশন পয়দা করব। ততক্ষণ হ্যাপি এরর হ্যান্ডলিং!

## Assertions

পাইথনে assertion তথা স্যানিটি চেক এনাবেল বা ডিজ্যাবল করে প্রোগ্রাম টেস্টিং এর কাজ করা হয়। কিন্তু, স্যানিটি চেক (sanity-check) আসলে কি? খুব দ্রুত একটি স্টেটমেন্টকে পর্যবেক্ষণ করে সেটার ফলাফলের সত্যতা যাচাই করাকেই স্যানিটি চেক বলা হয়।

assert স্টেটমেন্ট ব্যবহার করে এই কুইক টেস্ট করা হয়। যখন পাইথন কোন প্রোগ্রামের যেকোনো যায়গায় এই assert স্টেটমেন্টটি পায় তখন সেটাকে দ্রুত যাচাই করে এবং স্টেটমেন্টটি সত্য হোক সেটা আশা করে। কিন্তু তা না হলে পাইথন AssertionError টাইপের এক্সেপশন থ্রো (তৈরি) করে। একটি উদাহরণ দেখি -

*#print(1)*

**assert** 2 **+** 2 **==** 4

*#print(2)*

**assert** 1 **+** 1 **==** 3

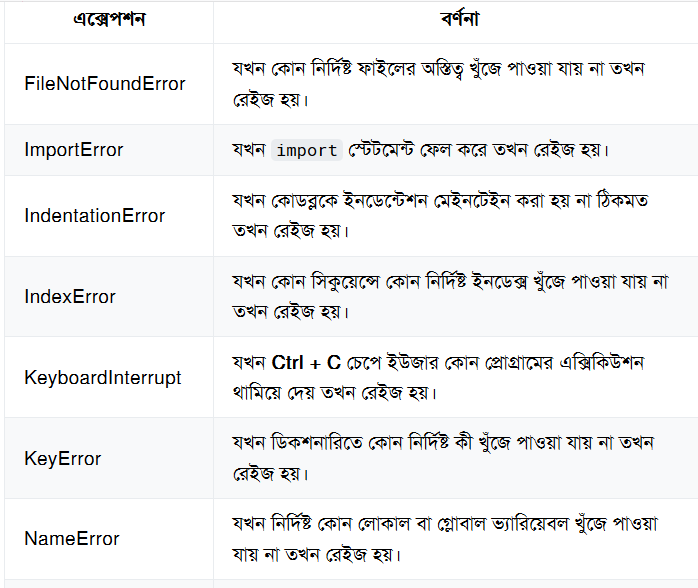
*#print(3)*

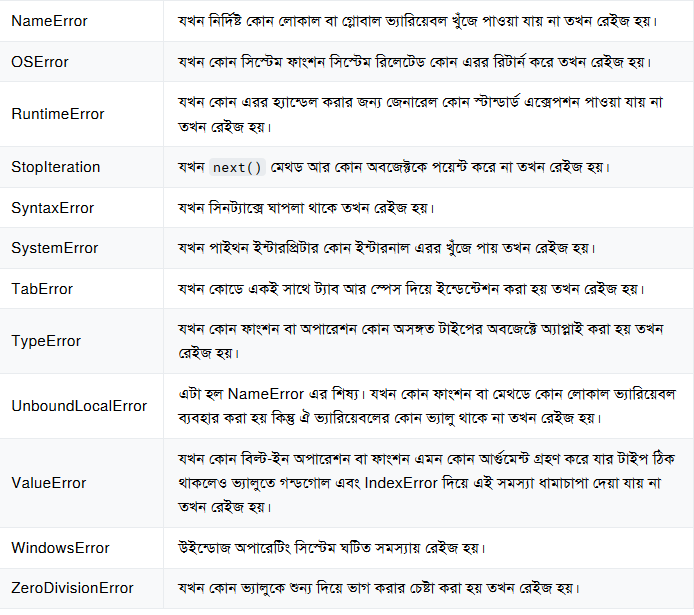
উপরের প্রোগ্রামের প্রথম প্রিন্ট স্টেটমেন্টের পর একটি assertion সেট করা হয়েছে। সেখানে একটি সাধারণ অ্যারিদম্যাটিক কন্ডিশন যাচাই করা হয়েছে assert ব্যবহার করে। সেই স্যানিটি চেকটি সত্য বা পাশ হয়েছে (২ আর ২ যোগ করলে ৪ হয়)। তাই, print(2) স্টেটমেন্ট কাজ করছে। এরপর আবার একটি স্যানিটি চেক সেট করা হয়েছে। কিন্তু, স্বাভাবিক ভাবেই সেটি সত্য নয় (১ আর ১ যোগ করে ৩ হয় না)। তাই পাইথন সেখানে একটি AssertionError এক্সেপশন থ্রো করেছে। আর তাই, এর পরে থাকা print(3) স্টেটমেন্টটি এক্সিকিউটও হয় নি।

সাধারণত প্রোগ্রামারগণ কোন একটি ফাংশনের ডেফিনেশনের শুরুতেই এরকম স্যানিটি চেক ব্যবহার করেন ইনপুট/আর্গুমেন্ট ডাটা চেক করার জন্য। আবার ফাংশন কল এর পরেও ব্যবহার করে থাকেন ফাংশনের আউটপুট ডাটা চেক করার জন্য।

বলা বাহুল্য, অন্যান্য এক্সেপশনের মত এই এক্সেপশনকেও try, except দিয়ে হ্যান্ডেল করা যায়।

## Different error and হ্যান্ডলিং





## রেগুলার এক্সপ্রেশন

Regular Expressions বা regex বা regexp হচ্ছে কিছু কারেকটারের সিকোয়েন্স, যা একটা সার্চ প্যাটার্ন তৈরি করে। পুরা লাইনটার মধ্যে সব গুলোই ইংরেজী শব্দ। বাংলা করলে ভয়াবহ হবে। তাহলে ইংরেজীতেই দেখি রেগুলার এক্সপ্রেশন কিঃ

A regular expression is a sequence of characters that forms a search pattern.

যে সার্চ প্যাটার্ণটি তৈরি করে, তা কোন টেক্সট সার্চ করার জন্য অথবা কোন টেক্সট রিপ্লেস করার জন্য ব্যবহৃত হয়। একটা উদারহণ দেওয়া যাক। ধরা যাক আমাদের বিশাল একটা টেক্সট ফাইল দেওয়া হল। যেখানে অনেক গুলো লেখার মাঝে মাঝে ইমেইল ও রয়েছে। আমাদের শুধু ইমেইল গুলো বের করতে হবে। ধরা যাক এক লক্ষ ইমেইল রয়েছে ঐ বিশাল টেক্সট ফাইলের মধ্যে। এখন কাউকে যদি ইমেইল গুলো খুজে বের করতে বলে, তাহলে অনেক সময় লাগবে। আবার অনেক গুলো ইমেইল সে হয়তো চোখের ভুলের কারণে মিস করে যাবে। আমাদের দরকার অটোমেটিক কিছু। আর এ কাজ টাই করে রেগুলার এক্সপ্রেশনে। আমাদের আগে একটা সার্চ প্যাটার্ণ তৈরি করতে হবে। ইমেইল এড্রেসে কি কি থাকে সে গুলো এনালাইসিস করে। যেমন একটা ডকুমেন্ট থেকে ইমেইল গুলো বের করার সার্চ প্যাটার্ণ হতে পারে নিচের মতঃ

b[A-Z0-9.\_%+-]+@[A-Z0-9.-]+.[A-Z]{2,4}b

এই সার্চ প্যাটার্ণ ব্যবহার করে আমরা সহজেই সকল ইমেইল খুজে বের করে ফেলতে পারব। এখন এক একটা প্রোগ্রামিং ল্যাঙ্গুয়েজে এক এক ভাবে এই রেগুলার এক্সপ্রেশন ব্যবহার করতে হয়।  কি সব লিখছি, তাই তো? একবার বুঝলেই সহজ হয়ে যাবে। উপরের এটা একটা উদারহণ, আমরা আরো ছোট খাটো উদাহরণ দিয়ে দেখব। তারপর কিভাবে কমপ্লেক্স সার্চ প্যাটার্ন তৈরি করতে পারি, তা দেখব।

রেগুলার এক্সপ্রেশন নিয়ে কাজ করার জন্য রেগুলার এক্সপ্রেশন মডিউলটা ইম্পোর্ট করে নিতে হবে। তার জন্য import re যুক্ত করে নিতে হবে।

ছোট বেলায় সবাই পড়ছি The Lion and the Mouse এর গল্পটি। আমরা যে ফোল্ডারে পাইথন প্রোগ্রাম লিখছি, সেখানে file.txt নামে একটি ফাইল তৈরি করি। ঐ ফাইলে আমাদের রেগুলার এক্সপ্রেশন নিয়ে কাজ করার জন্য গল্পটি file.txt এ রাখিঃ

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | Once when a Lion was asleep, a little Mouse began running up and down upon him.    This soon wakened the Lion, who placed his huge paw upon him and opened his big jaws to swallow him.    "Pardon, O King!" cried the little Mouse, "Forgive me this time.    I shall never repeat it and I shall never forget your kindness. And who knows, but I may be able to do you a good turn one of these days?"    The Lion was so tickled at the idea of the Mouse being able to help him, that he lifted up his paw and let him go.    Short Stories Sometime later a few hunters captured the King and tied him to a tree while they went in search of a wagon to carry him on.    Just then the little Mouse happened to pass by, and seeing the sad plight in which the Lion was, ran up to him and soon gnawed away the ropes that bound the King of the Beasts. "Was I not right?" said the little Mouse, very happy to help the Lion. |

নিচের প্রোগ্রামটি দেখিঃ

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | import re    file = open('file.txt')    for line in file:      if re.search( '(waken|open)ed', line):          print (line) |

এখানে ফাইলটি ওপেন করা হয়েছে। এরপর এর প্রতিটি লাইনের জন্য wakened এবং opened দ দুইটা শব্দ সার্চ করা হয়েছে। “|” মানে হচ্ছে অথবা। OR

wakened এবং opened এ শেষে ed কমন রয়েছে। তাই আমরা দুইটা শব্দকে এভাবে লিখতে পারিঃ (weken অথবা open)ed, কম নেওয়ার মত। আর রেগুলার এক্সপ্রেশনে এটা লিখে এভাবেঃ (waken| open)ed  
আর উপরের প্রোগ্রামে যখন এ দুইটি ওয়ার্ডের যে কোন একটা পাবে, তাহলে ঐ লাইনটি প্রিন্ট করবে। আমাদের উপরের গল্পএ wakened এবং opened দুইটি একই লাইনে। তাই একটা লাইনই প্রিন্ট করবে।

আমরা কোন স্ট্রিং গ্রুপ ম্যাচ করে কিনা, তা চেক করতে পারি এভাবেঃ

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | import re    file = open( 'file.txt')    for line in file:      match = re.search( '(waken|happen)ed ' , line)      if match :          print ( match.group()) |

আমরা যদি একটা স্ট্রিং সার্চ করে তা রিপ্লেস করতে চাই, তাহলে লিখব এভাবেঃ

re.sub মেথড দিয়ে স্ট্রিং প্যাটার্ন সার্চ এবং রিপ্লেস করা হয়।

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | import re    file = open( 'file.txt')    for line in file:      print (re.sub( 'Lion' ,'Tiger' ,line)) |

# ফাইল

## ফাইল খোলা

প্রোগ্রামিং মানেই ফাইল থেকে ডাটা নিয়ে কাজ করা খুবই স্বাভাবিক একটি ঘটনা। আর পাইথনে ফাইল নিয়ে কাজ করা অনেক সহজ। এর মাধ্যমে কোন ফাইল থেকে ডাটা পড়া, ফাইলে নতুন ডাটা লেখা বা ফাইল কন্টেন্টকে আপডেট করা ইত্যাদি করা যায় খুব সহজে অল্প কোড লিখেই।

ফাইল নিয়ে কাজ করার শুরুতেই পাইথনের বিল্ট ইন ফাংশন open ব্যবহার করে সেই ফাইলকে ওপেন করে নিতে হবে। ওপেন মানে কোন এডিটরে ওপেন নয়, বরং পাইথনের কাছে সেটা ওপেন হয়ে থাকে কাজ করার উপযোগী মোডে। নিচের মত একটি লাইন লিখেই সেটি করা যায় -

file\_to\_work\_on **=** open("file\_name.txt")

open ফাংশনের আর্গুমেন্ট হিসেবে আলোচ্য ফাইলের পাথ দিতে হয়। যদি পাইথন স্ক্রিপ্ট এবং ফাইলটি কম্পিউটারের একই লোকেশনে থাকে তাহলে উপরের মত শুধু ফাইলের নামটি লিখলেই কাজ শেষ। না হলে file\_to\_work\_on = open("/Users/nuhil/Desktop/file\_name.txt") এভাবে লিখতে হতে পারে।

open ফাংশনের আরও কিছু আর্গুমেন্ট আছে যেমন - দ্বিতীয় আর্গুমেন্ট পাস করে নির্ধারণ করা হয়, পাইথন উক্ত ফাইলটিকে কোন মোডে খুলবে অর্থাৎ শুধু সেটি থেকে ডাটা পড়ার জন্য নাকি, সেখানে ডাটা লেখার জন্য নাকি নতুন ডাটা যুক্ত করার জন্য।

যেমন লেখার জন্য তথা রাইট মোডে খোলার জন্য -

file\_to\_work\_on **=** open("file\_name.txt", "w")

রিড মোডে খোলার জন্য –

file\_to\_work\_on **=** open("file\_name.txt", "r")

অ্যাপেন্ড তথা ফাইলের শেষে নতুন কন্টেন্ট যুক্ত করার জন্য সেই মোডে খুলতে –

file\_to\_work\_on **=** open("file\_name.txt", "a")

টেক্সট ফাইল নয় এমন ফাইল নিয়ে কাজ করার জন্য বাইনারি মোডে সেই ফাইলকে খুলতে হবে। যেমন একটি বাইনারি ফাইলকে রাইট মোডে খোলার জন্য –

file\_to\_work\_on **=** open("my\_file", "wb")

ফাইল খোলার পর সেটি নিয়ে কাজ শেষে গুরুত্বপূর্ণ আরেকটি টাস্ক হচ্ছে সেই ফাইলকে ক্লোজ বা বন্ধ করা। নাহলে অকারণেই পাইথনের কাছে ফাইলটি ওপেন অবস্থাতেই থাকবে যা বস্তুত মেমোরি দখল করে থাকবে এবং প্রোগ্রামের পারফর্মেন্সে খারাপ ভূমিকা রাখবে। মোট কথা, আমরা যেমন কম্পিউটারে কোন এডিটর দিয়ে একটি ফাইল খুলে সেখানে কাজ শেষে বন্ধ করি অযথা র‍্যাম নষ্ট না করার জন্য। একই কারনে প্রোগ্রাম্যাটিক্যালি কোন একটি ফাইল নিয়ে কাজের শেষেও সেটা বন্ধ করা উচিৎ। সিম্পল ব্যাপার, তাই না?

নিচের মত করে ক্লোজ করার কাজটি করা যায় -

file\_to\_work **=** open("filename.txt", "w")

*# do HERE whatever you like, with the file*

*# such as write new lines in it*

*# then close it*

file\_to\_work.close()

alla type example

*#read a file ,means previus created file to read mode*

sos**=**open('cool.txt','r')

sos.close()

*#write a file,means create a new file,same name er previus file takle setar all document erase hoye jabe*

sos**=**open('cool.txt','w')

sos.close()

*#append/add new data in file,means add new some with privus line*

sos**=**open('cool.txt','a')

sos.close()

*#write any type file, by binary mode,create any file without any extention*

sos**=**open('cool','wb')

sos.close()

## ফাইল পড়া

lol**=**open('cool.txt','r')

sos**=**lol.read()

print(sos)

lol.close()

প্রথমেই ওপেন ফাংশন ব্যবহার করে এবং ফাইলের পাথ ডিফাইন করে দিয়ে একটি ফাইল অবজেক্ট পেয়েছি lolনামের। এরপর এই অবেজক্টের মেথড read ব্যবহার করে পুরো ফাইলে থাকা কন্টেন্ট পড়ে sos ভ্যারিয়েবলে জমা করেছি। অতঃপর, একটি প্রিন্ট স্টেটমেন্ট ব্যবহার করে সেই কন্টেন্ট স্ক্রিনে প্রিন্ট করেছি। আর কাজ শেষে, ফাইল অবজেক্ট এর close মেথড ব্যবহার করে ফাইলকে ক্লোজ করেছি।

এভাবে পুরো কন্টেন্ট একসাথে না পড়ে বাইট হিসেবেও পড়া যায়। read মেথডের আর্গুমেন্ট হিসেবে কত বাইট পড়তে চাই সেটা পাঠিয়ে দেয়া যায়। উপরের প্রোগ্রামের একটু মডিফায়েড ভার্সন দেখি

lol**=**open('cool.txt','r')

sos**=**lol.read(5)

print(sos)

lol.close()

example

lol**=**open('cool.txt','r')

sos**=**lol.read(1)

tos**=**lol.read(2)

mos**=**lol.read(3)

hos**=**lol.read(5)

print(sos)

print(tos)

print(mos)

print(hos)

lol.close()

file\_to\_work = open("Test.txt", "r")

just\_one\_character = file\_to\_work.read(1)

print(just\_one\_character)

remaining\_four\_characters = file\_to\_work.read(4)

print(remaining\_four\_characters)

rest\_of\_the\_file = file\_to\_work.read()

print(rest\_of\_the\_file)

file\_to\_work.close()

উপরের প্রোগ্রামে তিন বার ফাইল থেকে কন্টেন্ট পড়া হয়েছে, কিন্তু তিনভাবে। প্রথমবার মাত্র একটি বাইট পড়া হয়েছে। এক বাইট মানে একটি ক্যারেক্টার। তাই সেটি প্রিন্ট করেছে শুধু H. এর পরে আবার পড়া হয়েছে ৪টি বাইট। তাই ello এই চার ক্যারেক্টার পড়া হয়েছে। যেহেতু আমরা একই ফাইল অবজেক্ট (file\_to\_work) নিয়ে দ্বিতীয় বারও কাজ করেছি তাই এইবার যে ৪বাইট পড়তে চেয়েছি সেটা আসলে H এর পর থেকে ৪বাইট। তৃতীয় বার কোন আর্গুমেন্ট ছাড়া read মেথড ব্যবহার করা হয়েছে এবং ফাইলের বাকী সব কন্টেন্ট পড়ে প্রিন্ট করা হয়েছে। এবারও যেহেতু একই ফাইল অবজেক্ট এর উপরেই কাজ করা হয়েছে তাই rest\_of\_the\_file ভ্যারিয়েবলে কিন্তু H, ello এর পর থেকে অর্থাৎ World ... থেকে শেষ পর্যন্ত সব কন্টেন্ট জমা হয়েছে।

ইতোমধ্যে অনেকের মনে হতে পারে, এভাবে পুরো কন্টেন্ট একবারে পড়া এবং সেগুলো নিয়ে কাজ করা একটু ঝামেলা হবে; তাদের জন্য আছে readlines মেথড। এই মেথড ব্যবহার করলে ফাইলের প্রত্যেকটি লাইন আলাদা আলাদা করে নিয়ে পাইথন একটি লিস্ট বানায় এবং লিস্টের এক একটি এলিমেন্ট হয় এক একটি লাইন। নিচের উদাহরণটি দেখি -

lol**=**open('cool.txt','r')

sos**=**lol.readline(5)

print(sos)

lol.close()

file\_to\_work = open("Test.txt", "r")

lines = file\_to\_work.readlines()

print(lines)

file\_to\_work.close()

অনেকেই হয়তো ভাবছেন লিস্ট যেহেতু পেয়ে গেছি তাহলে এবার লাইন বাই লাইন নিয়ে কাজ করার জন্য ফর লুপ ব্যবহার করে সহজেই কাজ করে ফেলবো। আপনার কথা মাথায় রেখেই পাইথনের ফর লুপ রেডি হয়েই আছে। নিচের উদাহরণটি দেখুন lol**=**open('cool.txt','r')

**for** sos **in** lol:

*#sos=lol.readline(5)*

    print(sos)

lol.close()

দেখুন কিভাবে আলাদা করে read বা readlines মেথড ব্যবহার না করেই সরাসরি ফর লুপ ব্যবহার করে প্রত্যেকটি লাইনকে অ্যাক্সেস করা যায়। আউটপুট স্ক্রিনে একটা করে ফাকা লাইন বেশি প্রিন্ট হয়েছে। এতে প্রমাণিত হয় যে, ফর লুপের মধ্যে থাকা প্রিন্ট স্টেটমেন্ট আলাদা আলাদা ভাবে তিনবার এক্সিকিউট হয়েছে যার কারনে প্রত্যেকবার প্রিন্টের পর একটি করে ফাকা লাইন প্রিন্ট হয়েছে।

## ফাইলে লেখা

*#this section cretae and write a line*

lol**=**open('cool.txt','w')

lol.write('i am lamyaa and nowsin')

lol.close()

*#this section read the file*

lol**=**open('cool.txt','r')

sos**=**lol.read()

print(sos)

lol.close()

উপরোক্ত প্রোগ্রামের দুটি অংশ। প্রথম অংশে ফাইলকে ওপেন করে সেখানে একটি লাইন লেখা হয়েছে। আমাদের চলতি উদাহরণ মোতাবেক এই নামের ফাইলটি আগে থেকেই ছিল। কিন্তু w মোডে খোলার কারনে এবং এখানে নতুন করে লেখার কারনে ওই ফাইলের আগের সব কন্টেন্ট মুছে যাবে এবং নতুন write করা কন্টেন্ট লেখা হবে। যদি ওই নামের ফাইল না থাকতো, তাহলে পাইথন নতুন করে ওই নামে একটি ফাইল তৈরি করে সেখানে লিখতো। লেখা শেষে ফাইলটিকে ক্লোজ করা হয়েছে। দ্বিতীয় অংশে আবার সেই ফাইলকে পড়ার জন্য r মোডে খোলা হয়েছে এবং সব কন্টেন্ট পড়ে স্ক্রিনে প্রিন্ট করা হয়েছে।

চাইলে ফাইল লেখার কাজ সফল হল কিনা এবং কি পরিমাণ কন্টেন্ট ফাইলে লেখা হল সেটা যাচাই করার জন্য write মেথডের রিটার্ন ভ্যালুকে ক্যাপচার করে দেখা যেতে পারে নিচের মত করে

*#this section cretae and write a line*

lol**=**open('cool.txt','w')

sos**=**lol.write('i')

**if** sos:

    print(sos)

lol.close()

## ফাইল নিয়ে সঠিক কাজ

ইতোমধ্যে বেশ কয়েকবার বলা হয়েছে যে, ফাইল নিয়ে কাজ শেষে সেটিকে ক্লোজ করা খুব দরকারি। তো, এই দরকারি কাজটা যাতে বার বার ভুল হয়ে না যায় এর জন্য কিছু টেকনিক অবলম্বন করা যেতে পারে বা অভ্যাসে পরিণত করা যেতে পারে। যেমন, নিচের প্রোগ্রামটি দেখি –

*#this section cretae and write a line*

**try**:

    lol**=**open('cool.txt','w')

    sos**=**lol.write('i')

**if** sos:

        print(sos)

**finally**:

    lol.close()

*#this section read the file*

**try**:

    lol**=**open('cool.txt','r')

    sos**=**lol.read()

    print(sos)

**finally**:

    lol.close()

মনে আছে, আমরা কয়েক চ্যাপ্টার আগেই finally ব্লক নিয়ে আলোচনা করেছি? try, except এর সাথে finally ব্লকের ব্যবহার আমরা দেখেছি এবং জানি যে এই ব্লকের মধ্যে যাই থাকুক না কেন, সেই কোড গুলো রান করবেই এমনকি যদি এর উপরের try, except ব্লকে অনাকাঙ্ক্ষিত কিছু ঘটেও। এটাই একটা টেকনিক, ফাইল ক্লোজ করতে ভুল না করার। উপরের প্রোগ্রামে আমরা ট্রাই ব্লকের মধ্যে ফাইল ওপেন এবং পড়ার কাজ করেছি এবং ফাইনালি ব্লকের মধ্যে ক্লোজ করেছি। এতে করে, ঘটনা যাই হোক, ফাইল ক্লোজ হবেই।

আরও একটি বেস্ট প্র্যাকটিস আছে। with স্টেটমেন্টের ব্যবহার। প্রথমে একটি উদাহরণ দেখি তারপর বিশ্লেষণ করা যাবে -

**with** open("Test.txt") **as** f:

    print(f.read())

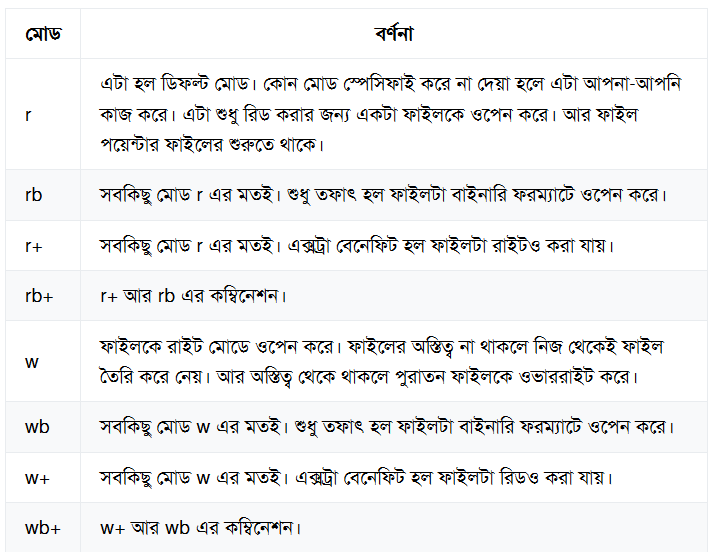
with স্টেটমেন্ট আসলে একটি টেম্পোরারি ভ্যারিয়েবল তৈরি করে। উপরের প্রোগ্রামে এটি ব্যবহার করে open("Test.txt") স্টেটমেন্টটির জন্য একটি টেম্পোরারি ভ্যারিয়েবল তৈরি করা হয়েছে f নামে। অর্থাৎ বস্তুত এমন হয়েছে f = open("Test.txt"). এই f কে with এর আওতাভুক্ত কোডে অর্থাৎ এর স্কোপে ব্যবহার করা যায়। আবার, with ব্যবহারের আরেকটি মজার ব্যাপার হচ্ছে এর আওতাভুক্ত কোড ব্লকের কাজ শেষ হয়ে গেলেই এর দ্বারা তৈরি টেম্পোরারি ভ্যারিয়েবলও ডেস্ট্রয় হয়ে যায়। এতে করে আমাদের উদ্দেশ্য হাসিল হয় তথা ফাইল ক্লোজের কাজটি হয়ে যায়। এখন পর্যন্ত এটাকেই ফাইল নিয়ে ছোট খাটো কাজ করার বেস্ট প্র্যাকটিস হিসেবে ধরা হয়।

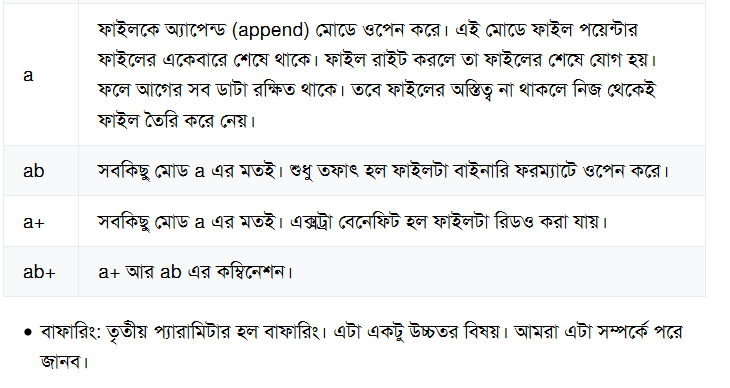
## Different file mode

ফাইল নিয়ে কাজ করতে গিয়ে আমরা তিনটা জিনিস শিখব। ফাইল ওপেন (open) করা, ওপেন করা ফাইল রিড (read) করা, রিড করার পর ফাইল রাইট (write) করা আর শেষমেষ ফাইল ক্লোজ (close) করা।

পাইথনে ফাইল ওপেন করার জন্য open() ফাংশনটা ব্যবহার করব আমরা। এই ফাংশনটা তিনটা প্যারামিটার নিয়ে কাজ করে।

* ফাইলের নাম: প্রথম প্যারামিটার হল ফাইলের নাম। আমাদের স্ক্রিপ্ট ও আলোচ্য ফাইল যদি একই ডিরেক্টরিতে হয় তবে শুধু ফাইলের নামটা স্ট্রিং হিসাবে দিতে হয়। ফাইল যদি আলাদা ডিরেক্টরিতে থাকে তবে পুরো পাথটা স্ট্রিং হিসাবে দিতে হবে।
* অ্যাক্সেস মোড: দ্বিতীয় প্যারামিটার হল অ্যাক্সেস মোড। একটা ফাইলকে আমরা বিভিন্ন কাজের জন্য ওপেন করতে পারি। সেক্ষেত্রে ভিন্ন ভিন্ন কাজের জন্য ভিন্ন ভিন্ন মোডে ফাইলটা ওপেন করতে হবে। নিচে কতগুলো মোডের নাম ও কোন কাজে লাগে তা দেয়া হল।





Example:

my\_file = open('test.txt', 'r')

content = my\_file.read()

print(content)

my\_file.close()

প্রথম লাইনে আমরা ফাইলটা রিড মোডে ওপেন করেছি। দ্বিতীয় লাইনে ফাইলটা read() ফাংশনের (আসলে মেথড) সাহায্যে রিড করে ডাটা content ভ্যারিয়েবলে অ্যাসাইন করেছি। আর শেষ লাইনে close() ফাংশন (মেথড) ব্যবহার করে ফাইলটা ক্লোজ করে দিয়েছি। আরেকটা প্রোগ্রাম দেখা যাক।

my\_file = open('test.txt', 'w')

my\_file.write('I am Maksudur Rahman Maateen.')

my\_file.close()

এখানে write() ফাংশন (মেথড) ব্যবহার করে আমরা ফাইলটা রাইট করেছি। এই ফাংশনের ভিতর স্ট্রিং হিসাবে যা দেয়া হবে তাই রাইট হবে ফাইলে। প্রোগ্রামটা রান করলে আমরা আউটপুট কিছুই পাব না। তবে ফাইলটা যদি নরমালি ওপেন করে দেখি তাহলে দেখব সেখানে এখন The name of my country is Bangladesh. এর পরিবর্তে I am Maksudur Rahman Maateen. লেখা রয়েছে। এবার আরেকটা প্রোগ্রাম দেখা যাক।

my\_file = open('test.txt', 'a')

my\_file.write('I am from Bangladesh.')

my\_file.close()

এই প্রোগ্রামটা রান করলেও আমরা আউটপুট কিছুই পাব না। তবে ফাইলটা যদি নরমালি ওপেন করে দেখি তাহলে দেখব সেখানে এখন I am Maksudur Rahman Maateen. এর পরে I am from Bangladesh. লেখা রয়েছে। এবার আরেকটা প্রোগ্রাম দেখা যাক।

my\_file = open('test.txt', 'r')

content = my\_file.read(5)

print(content)

content = my\_file.read()

print(content)

position = my\_file.tell()

print(position)

my\_file.seek(0, 0)

content = my\_file.read()

print(content)

my\_file.close()

লম্বা-চওড়া একটা প্রোগ্রাম। আমরা এবার এটা ব্যাখায় যাব। দ্বিতীয় লাইনে read() এর ভিতর আমরা 5 ভ্যালু পাস করেছি। এর ফলে ফাইলের প্রথম ৫ টা ক্যারেক্টার পর্যন্ত রিড হবে শুধু। একটা কথাটা তো আমরা নিশ্চয়ই জানি, স্পেসও একটা ক্যারেক্টার। যাহোক, ষষ্ঠ লাইনে এসে আমরা tell() ফাংশন দিয়ে ফাইল পয়েন্টারের পজিশন খুঁজে বের করেছি। আর অষ্টম লাইনে seek() ফাংশন দিয়ে ফাইল পয়েন্টার আবার শুরুতে নিয়ে গিয়েছি। আরেকটা উদাহরণ দেখা যাক।

with open('test.txt', 'r') as my\_file:

content = my\_file.read()

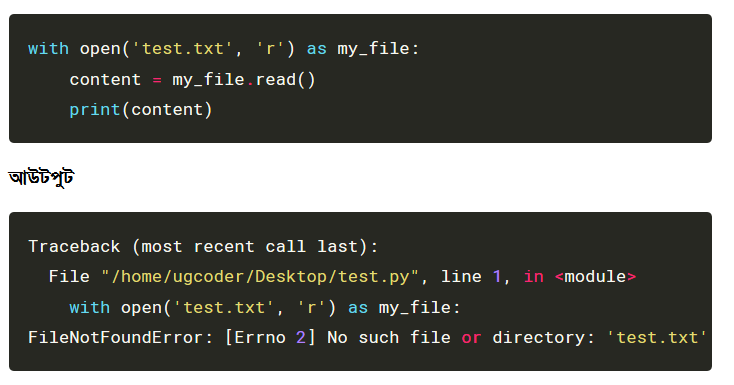
print(content)

এই প্রোগ্রামে আমরা with স্টেটমেন্ট ব্যবহার করেছি। স্টেটমেন্টটা ব্যবহার করে ফাইলটা my\_file নামে ওপেন করেছি। বাদবাকি অপারেশন নরমাল। কেউ কি লক্ষ্য করেছি যে আমরা ফাইলটা ক্লোজ করিনি। আসলে এটাই এই স্টেটমেন্টের সুবিধা। কাজ শেষে বা কোন এরর পেলে নিজ থেকেই ফাইলটা ক্লোজ করে দেয়।

এই ছিল ফাইল নিয়ে আমাদের বেসিক আলোচনা। এখন বাড়ির কাজ আছে। একটা ফাইলে লুপ ঘুরিয়ে ১ থেকে ১০০ পর্যন্ত লাইন বাই লাইন রাইট করব আমরা। কি পারব তো সবাই? চেষ্টা করলে অবশ্যই পারব।

## এক্সেপশন in file

ফাইল চাপ্টারে আমরা সবার শেষে যে প্রোগ্রামটা লিখেছিলাম সেটা আবার চালাব আমরা। তবে এবার test.txt ফাইলটা ঐ জায়গা থেকে ডিলিট করে দেয়ার পর।



test.txt ফাইলটা না পেয়ে FileNotFoundError থ্রো করেছে পাইথন, তারপর প্রোগ্রামের এক্সিকিউশন বন্ধ হয়ে গেছে। ফলে শেষের নির্ভেজাল স্টেটমেন্টটাও এক্সিকিউট হয় নাই। এইরকম সমস্যার নিরসন করতেই এক্সেপশন হ্যান্ডলিং বা এরর হ্যান্ডলিং।

## try … except

উপরের এররটাই এবার আমরা হ্যান্ডেল করব। আগে আমরা প্রোগ্রামটা দেখব তারপর ব্যাখ্যায় যাব।

try:

with open('test.txt', 'r') as my\_file:

content = my\_file.read()

print(content)

except:

print('The file does not exist.')

print('Made by Maateen.')

এবার কিন্তু আর প্রোগ্রামটা ঠুস করেই শেষ হয়ে যায়নি। এররটা হ্যান্ডেল হয়ে শেষ স্টেটমেন্ট অবধি সুন্দরভাবে প্রোগ্রাম এক্সিকিউট হয়েছে।

try...except এর ব্যাপারটা হল এরকম: নরমাল কোডগুলো try ব্লকের ভিতর থাকবে। আর কোন এক্সেপশন রেইজ হলে except ব্লকের কোড এক্সিকিউট হবে। কোন নির্দিষ্ট এক্সেপশন হ্যান্ডেল করার জন্য কোড লিখতে চাইলে except এর পরে স্পেস দিয়ে এক্সেপশনের নাম দিয়ে দিতে হয়। আবার চাইলে এক্সেপশনের থ্রো করা মেসেজ হোল্ড করে প্রিন্ট করে ইউজারকে দেখানোও যায়। একটা উদাহরণ দেখা যাক।

**with** open('cool.txt','r') **as** r:

    lol**=**r.read()

    print(lol)

example:

**try**:

**with** open('coo.txt','r') **as** r:

        lol**=**r.read()

        print(lol)

**except**:

    print('can not find file')

example:

**try**:

**with** open('test.txt', 'r') **as** my\_file:

        content **=** my\_file.read()

        print(content)

**except** FileNotFoundError:

    print('The file does not exist.')

print('Made by Maateen.')

**try**:

    my\_list **=** []

    print(my\_list[0])

**except** IndexError **as** e:

    print(e)

এখানে আমরা এক্সেপশনের নাম ধরে এরর হ্যান্ডেল করেছি। শেষের এক্সেপশনে পাইথনের জেনারেট করা এরর মেসেজই প্রিন্ট করে দিয়েছি। আচ্ছা, একটা try ব্লকের কোডে কয়েকটা এরর হতে পারে। সেক্ষেত্রে কিভাবে এরর হ্যান্ডেল করব আমরা?

আসলে একটা try ব্লকের জন্য যত খুশি তত except ব্লক লেখা যায়। তবে প্রতিটা except ব্লকে এক্সেপশনের নাম উল্লেখ করতে হবে। একটা নির্দিষ্ট এক্সেপশন রেইজ হলেই কেবল ঐ নির্দিষ্ট except ব্লক এক্সিকিউট হবে। একটা উদাহরণ দেখা যাক।

try:

my\_file = open('test.txt')

content = my\_file.read()

i = int(content.strip())

except IOError as e:

errno, strerror = e.args

print("I/O error({0}): {1}".format(errno,strerror))

except ValueError:

print("No valid integer in line.")

except:

print("Unexpected error!")

**আউটপুট**

I/O error(2): No such file or directory

এখানে আমরা একটা try ব্লকের জন্য তিনটা except ব্লক লিখেছি। প্রথমটা IOError এর জন্য, দ্বিতীয়টা ValueError এর জন্য আর শেষেরটা ঐ দুইটা বাদে যেকোন এররের জন্য। এ ধরনের except কে পাইথনিক ভাষায় Bare Except বলা হয়। বাংলায় অবশ্য আমরা এর একটা নাম দিতে পারি - ল্যাংটা এক্সেপ্ট। আর পারতপক্ষে ল্যাংটা এক্সেপ্ট স্কিপ করা উচিত আমাদের। কারণ, বেয়ার এক্সেপ্ট সব ধরনের এররকে হাইড করে দেয় ফলে আমরা জানতেই পারব না ঠিক কোন এক্সেপশনটাকে আমরা ক্যাচ করছি। তাই দেশ ও জাতির বৃহত্তর কল্যাণের স্বার্থে আমরা সবসময় এক্সেপশনের নাম উল্লেখ করে এরর হ্যান্ডেল করব।

আমরা চাইলে সবগুলো এররকে একটা except ব্লকে সেটেল করে দিতে পারতাম।

try:

my\_file = open('test.txt')

content = my\_file.read()

i = int(content.strip())

except (IOError, ValueError):

pass

## try … except … else

else ব্লক except ব্লকের শেষে বসে। try ব্লকে কোন এক্সেপশন রেইজ না হলেই কেবল else ব্লকের কোড এক্সিকিউট হয়। একটা উদাহরণ দেখা যাক।

**try**:

**with** open('cool.txt','r') **as** r:

        lol**=**r.read()

        print(lol)

**except** FileNotFoundError:

    print('can not find file')

**else**:

    print('i am runn if file have')

কোন এক্সেপশন রেইজ হয়নি। তাই else ব্লকের কোড এক্সিকিউট হয়েছে।

## try … except … finally

finally ব্লক একেবারে শেষে বসে। আর কোন এক্সেপশন রেইজ হোক আর নাই বা হোক, finally ব্লকের কোড ঠিকই এক্সিকিউট হয়। এজন্য একে ক্লীন-আপ অ্যাকশন বলা হয়। একটা উদাহরণ দেখা যাক।

**try**:

**with** open('coo.txt','r') **as** r:

        lol**=**r.read()

        print(lol)

**except** FileNotFoundError:

    print('can not find file')

**else**:

    print('i am runn if file have')

**finally**:

    print('i am must run, if raise occur or not')

test.txt ফাইলটা না থাকায় FileNotFoundError এক্সেপশন রেইজ হয়েছে। কিন্তু তারপরও finally ব্লকের কোড ঠিকই রেইজ হয়েছে।

# Extra/others topics

## (Gap) ভার্চ্যুয়াল এনভায়রনমেন্ট

প্রথমেই আমাদের জানা দরকার ভার্চ্যুয়াল এনভায়রনমেন্ট আসলে কী জিনিস। আমরা জানি, পাইথন একটা মডার্ন প্রোগ্রামিং ল্যাঙ্গুয়েজ। অন্যান্য মডার্ন প্রোগ্রামিং ল্যাঙ্গুয়েজের মতো বিভিন্ন প্যাকেজ এবং মডিউল হ্যান্ডেল করার মৌলিক পদ্ধতি রয়েছে পাইথনের। আমরা ইতিমধ্যেই pip এর সাথে পরিচিত হয়েছি। এর কথাই বলছিলাম আর কি!

ধরা যাক, আমাদের দুটি প্রজেক্ট আছে : A ও B; দুটি প্রজেক্টেই আমরা requests প্যাকেজটি ব্যবহার করব। প্যাকেজটি ইনস্টল করার জন্য নিচের কমান্ড চালালেই চলে:

sudo pip3 install requests

এই কমান্ড টার্মিনালে চালালে প্যাকেজটার 2.12.4 ভার্সন আমাদের পিসিতে ইনস্টল হয়ে যাবে। এই যে প্যাকেজটা ইনস্টল করলাম, এটা কিন্তু গ্লোবাললি ইনস্টল করলাম। কথাটা একটু শক্ত মনে হতে পারে যে গ্লোবাললি আবার কী রকম কথা! কিন্তু একটু পরেই আমরা কথাটার তাৎপর্য বুঝতে পারব।

requests প্যাকেজটার যে ভার্সন আমরা ইনস্টল করলাম তা দিয়ে প্রজেক্ট A-র কাজ খুব ভালোভাবেই চলবে। কিন্তু প্রজেক্ট B-র দরকার requests প্যাকেজের 1.2.1 ভার্সনটা। এখন যদি এই ভার্সনটা ইনস্টল করতে যাই তাহলে 2.12.4 ভার্সনটা ওভাররাইট হয়ে যাবে। অথচ প্রজেক্ট A-র জন্য ওই ভার্সনটাও দরকার আমাদের। কী উপায় এখন? যদি কোনোভাবে দুটি ভার্সনই ইনস্টল করে কাজ করা যেত তাহলে সব সমস্যার সমাধান হয়ে যেত। আর এসব সমস্যা থেকে চিরতরে মুক্তি পেতেই ভার্চ্যুয়াল এনভায়রনমেন্ট আইডিয়া তৈরি হয়েছে। তা ছাড়া সফটওয়্যার টেস্ট করতেও ভার্চ্যুয়াল এনভায়রনমেন্ট অনেক কাজে আসে।

পাইথনের অফিশিয়াল ডকে ভার্চ্যুয়াল এনভায়রনমেন্টের একটি সংজ্ঞায় বলা হয়েছে-

ভার্চ্যুয়াল এনভায়রনমেন্ট হলো এমন একটি পাইথন এনভায়রনমেন্ট যেখানে ইনস্টল করা পাইথন ইন্টারপ্রিটার, লাইব্রেরি ও স্ক্রিপ্ট অন্যান্য ভার্চ্যুয়াল এনভায়রনমেন্ট ও সিস্টেম পাইথনে ইনস্টল করা লাইব্রেরি থেকে আলাদা। মূলত ভার্চ্যুয়াল এনভায়রনমেন্ট একটি ডিরেক্টরি ট্রি যাতে পাইথন এক্সিকিউটেবল ফাইল ও অন্যান্য ফাইল থাকে, যা দ্বারা প্রকাশ পায় যে এটি ভার্চ্যুয়াল এনভায়রনমেন্ট ছাড়া অন্য কিছু নয়।

ভার্চ্যুয়াল এনভায়রনমেন্টের মধ্যে আমরা পাইথন ইন্টারপ্রিটার ও এর বিভিন্ন প্যাকেজ লোকাললি ইনস্টল করতে পারি। লোকাললি বলা হলো এ জন্য যে ওই এনভায়রনমেন্টের বাইরে ওই ইন্টারপ্রিটার ও প্যাকেজের কোনো প্রভাব থাকবে না। যেসব প্যাকেজের প্রভাব পুরো সিস্টেমজুড়ে থাকে তাদের গ্লোবাল প্যাকেজ বলা হয়। এই হলো গ্লোবাল আর লোকাল আইডিয়ার পার্থক্য।

ভার্চ্যুয়াল এনভায়রনমেন্ট তৈরি করার জন্য আমরা pyvenv ও virtualenv কমান্ডলাইন টুল দুটির যেকোনো একটি ব্যবহার করতে পারি। পাইথন 3.4 থেকে pyvenv টুলটি পাইথন স্ট্যান্ডার্ড লাইব্রেরির সাথেই পাওয়া হচ্ছে। তবে আমরা virtualenv থার্ড-পার্টি প্যাকেজটি ব্যবহার করব। এটি একটি পিওর-পাইথন ডিস্ট্রিবিউশন। আমরা এখন টুলটা ইনস্টল করব। সে জন্য টার্মিনালে কমান্ড দেব-

sudo pip3 install virtualenv

ইনস্টল হয়ে গেলে আমরা নতুন একটি প্রজেক্ট ডিরেক্টরি তৈরি করে তার মধ্যে লাফ দেব-

mkdir ~/Desktop/newproject

cd ~/Desktop/newproject

হয়ে গেল নতুন প্রজেক্ট ডিরেক্টরি। তাহলে এর মধ্যে ভার্চ্যুয়াল এনভায়রনমেন্ট তৈরি করার কাজকর্ম শুরু করে দেওয়া যাক-

virtualenv env

এখানে env মূলত ভার্চ্যুয়াল এনভায়রনমেন্ট ডিরেক্টরি। আমরা চাইলে এর বদলে যা খুশি তা-ই দিতে পারি। আমাদের তৈরি এই নতুন ভার্চ্যুয়াল এনভায়রনমেন্টকে অ্যাকটিভেট করার জন্য আমাদের টার্মিনালে কমান্ড চালাতে হবে-

source env/bin/activate

ওপরের কমান্ডটা চালালেই আমাদের ভার্চ্যুয়াল এনভায়রনমেন্ট চালু হয়ে যাওয়ার কথা। তবে আমরা যদি টার্মিনালে ব্যাস শেল বাদ দিয়ে, fish বা csh ব্যবহার করি সে ক্ষেত্রে ওপরের কমান্ড কাজ করবে না। তবে এ সমস্যার সহজ সমাধান আছে। ফিশের ক্ষেত্রে ওপরের কমান্ডের বদলে টার্মিনালে নিচের কমান্ড চালাব-

source env/bin/activate.fish

আর csh-এর ক্ষেত্রে ওপরের কমান্ডের বদলে টার্মিনালে নিচের কমান্ড চালাব-

source env/bin/activate.csh

আমাদের এই নতুন ভার্চ্যুয়াল এনভায়রনমেন্টে পাইথনের কোন ভার্সন চলছে তা একটু চেক করে দেখা যাক-

python -V

এই কমান্ড দিলেই রানিং পাইথনের ভার্সন শো করবে। এবার pip ব্যবহার করে requests প্যাকেজের 1.2.1 ভার্সনটি ইনস্টল করা যাক-

pip install -Iv requests==1.2.1

কিন্তু যদি লেটেস্ট ভার্সনই ইনস্টল করতে চাই তাহলে বরাবরের মতো সেই পুরোনো কমান্ডই দিতে হবে-

pip install requests

এই হলো ভার্চ্যুয়াল এনভায়রনমেন্ট নিয়ে যাবতীয় আলোচনা। এ ছাড়া virtualenvwrapper, virtualenv-burrito, autoenv ইত্যাদি টুল নিয়ে গুগলে সার্চ করে ভালোভাবে জানার সুযোগ আছে।

## ইউনিট টেস্টিং

প্রোগ্রামিং আর ইউনিট টেস্টিং একসূত্রে গাঁথা, নয়নের অংশ যেমন নয়নের পাতা।

আমরা তো ইতিমধ্যে ইউজার ডিফাইন্ড ফাংশন শিখে ফেলেছি। শুধু তাই নয়, ছোট ছোট অনেক ইউজার ডিফাইন্ড ফাংশন লিখে কাজও করেছি। এইসব ফাংশন ঠিকমত কাজ করে কিনা সেটা বোঝার জন্য কি করেছি আমরা? কি আবার! প্রোগ্রাম রান করে দেখেছি। আচ্ছা, যখন আমরা বড়সড় প্রজেক্ট করব যেখানে হাজার-হাজার ফাংশন থাকবে তখনও কি বারবার প্রোগ্রাম রান করে দেখব?

অবশ্যই না। আসলে সেটা তো সম্ভবও না। বড় বড় কোম্পানিতে কোন প্রোগ্রামারই সম্পূর্ণ প্রজেক্ট একা করেন না। বরং সবাই ছোট ছোট অংশের কাজ করে। সেক্ষেত্রে টেস্টিংয়ের জন্য আমাদেরকে বিকল্প উপায় খুঁজে বের করতে হবে।

ইউনিট টেস্টিং হল প্রোগ্রামের বিভিন্ন ফাংশনকে অটোম্যাটিকালি টেস্ট করার উপায়। এইক্ষেত্রে প্রতিটি ফাংশনের জন্য প্রোগ্রামের ভিতর টেস্ট কোড লেখা হয়। এই টেস্ট কোড লেখার জন্য বিভিন্ন ফ্রেমওয়ার্ক, টুলস ব্যবহার করা হয়। যেমন: unittest, py.test, nose, tox, unittest2, mock ইত্যাদি।

### unittest

এটা পাইথনের ডিফল্ট ইউনিট টেস্টিং ফ্রেমওয়ার্ক, স্টান্ডার্ড লাইব্রেরির অন্তর্গত। কোড টেস্ট করার মোটামুটি সব দরকারি জিনিসই এতে পাওয়া যায়। যাহোক, এখন আমরা hijibiji.py ফাইলে কয়েকটা ফাংশন লিখব।

def add(a, b):

return a + b

def is\_even(number):

if (number % 2) == 0:

return True

else:

return False

এই ফাংশন দুইটা টেস্ট করার জন্য আমরা একই ডিরেক্টরির test.py তে টেস্ট কোড লিখব।

import unittest

from hijibiji import add, is\_even

class MyTest(unittest.TestCase):

def test\_add(self):

self.assertEqual(add(2, 3), 5)

def test\_is\_even(self):

self.assertTrue(is\_even(2))

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

**আউটপুট**

..

----------------------------------------------------------------------

Ran 2 tests in 0.010s

OK

কি করলাম আমরা? একটু ব্যাখ্যা করা যাক। add() ফাংশনটা দুইটা ভ্যালুকে যোগ করে। তাই আমরা টেস্ট করে দেখেছি যে add() ফাংশনে 2, 3 পাস করলে রিটার্নকৃত মান সত্যিই 5 এর সমান হয় কিনা। আর is\_even() ফাংশনটা চেক করে দেখে যে কোন সংখ্যা জোড় কিনা! জেড় হলে True রিটার্ন করে, বিজোড় হলে False রিটার্ন করে। তাই আমরা টেস্ট করে দেখেছি যে is\_even() ফাংশনে 2 পাস করলে তা True রিটার্ন করে কিনা।

আসলে টেস্ট কোড এভাবেই লেখা হয়। এই মূুহুর্তে পুরো ফ্রেমওয়ার্কের বর্ণনা দেয়া তো সম্ভব না। আমরা যদি আরো অনেক কিছু জানতে চাই তবে সেটা অফিসিয়াল ডক ( <https://docs.python.org/3.5/library/unittest.html> ) থেকে শিখে নিতে হবে।

### pytest

pytest হল ম্যাচিউর, ফুল-ফিচার্ড থার্ট-পার্টি পাইথন টেস্টিং টুল। unittest এর চেয়ে অনেক বেশি সহজ ও ফিচারসমৃদ্ধ। ইন্সটল করার জন্য নিচের কমান্ডটা দিব:

sudo pip3 install -U pytest

ইন্সটল হয়ে গেল pytest। এবার আমরা test.py তে লেখা আমাদের টেস্ট কোড কে রি-রাইট করব।

from hijibiji import add, is\_even

def test\_add():

assert add(2, 3) == 5

def test\_is\_even():

assert is\_even(2) == True

এবার টার্মিনালে আমরা নিচের কমান্ডটি দেব।

pytest test.py

**আউটপুট**

============================= test session starts ==============================

platform linux -- Python 3.5.2, pytest-3.0.3, py-1.4.31, pluggy-0.4.0

rootdir: /home/ugcoder/Desktop, inifile:

collected 2 items

test.py ..

=========================== 2 passed in 0.03 seconds ===========================

আগের চেয়ে অনেক সহজে আমরা টেস্ট করে ফেললাম। pytest নিয়ে ডিটেইলস জানার জন্য আমার এর অফিসিয়াল ডক ( <http://docs.pytest.org/en/latest/index.html> ) ফলো করব।

### টেস্টিং আর কিছু নিয়ম-নীতি

টেস্টিংয়ের সময় কিছু নিয়ম-নীতি মানা উচিত।

(১) একটি টেস্ট ইউনিট শুধু একটি ছোট্ট ফাংশনালিটির ওপর ফোকাস করবে আর সেটাকে সঠিক প্রমাণ করবে।

(২) প্রতিটি টেস্ট ইউনিট ইনডিপেনডেন্ট হবে। মানে একা একাই রান করতে পারবে।

(৩) টেস্ট ইউনিট কোড এমনভাবে করতে হবে যেন তা খুব দ্রুত রান করে। একটা টেস্ট ইউনিট রান হতে হতেই যদি আরেকটি কোরবানির ঈদ চলে আসে তাহলে প্রজেক্ট শিকায় উঠবে।

(৪) আমাদের সিঙ্গেল টেস্ট ইউনিট রান করা শিখা উচিত। একটা ফাংশনে বারবার চেঞ্জ আনার সময়, প্রতিবার টেস্ট ইউনিট রান করে দেখতে হবে সবকিছু ঠিকঠাক আছে কি না।

(৫) প্রতিবার কোনো প্রজেক্টে কোড করতে শুরু করার আগে ও পরে টেস্ট ইউনিটগুলো রান করে দেখতে হবে।

হ্যাপি ইউনিট টেস্টিং!

## ডিবাগিং ও লগিং

### ডিবাগিং

যখন আমরা একটা প্রোগ্রাম লিখি, তখন সেখানে সাধারণত দুই ধরনের এরর থাকে। এগুলো হলো সিনট্যাক্স এরর ও লজিকাল এরর।

সিনট্যাক্স এরর প্রোগ্রাম চালালে সহজেই ধরা পড়ে। তবে লজিকাল এরর ধরাটা একটু কষ্টসাধ্য। ইউনিট টেস্টিংয়ের মাধ্যমে লজিকাল এরর অনেকটাই ধরা পড়ে। তবে তারপরও লজিকাল এরর ধরাটা খুব কঠিন। অনেক সময় দেখব সবকিছু ঠিকঠাক রেখে প্রোগ্রাম লিখলাম কিন্তু আউটপুট যা আসার কথা তা আসছে না। এই যে সমস্যাটা, এটাকেই আমরা প্রোগ্রামিংয়ের ভাষায় বাগ বলি। আর এই বাগ মারার উপায় হলো ডিবাগিং। যে অস্ত্রপাতি দিয়ে বাগ মারা হয় তাকে বলা হয় ডিবাগার।

ভাগ্য ভালোই বলতে হবে, পাইথনে একটা অস্থির রকমের বিল্ট-ইন ডিবাগার আছে। এর নাম pdb (পিডিবি) বা পুরো নামে ডাকলে Python Debugger (পাইথন ডিবাগার)। আমরা তিন ভাবে এই ডিবাগারটা দিয়ে কাজ করতে পারি।

১. পিডিবি মূলত একটি মডিউল। তাই আমরা আমাদের প্রোগ্রামে একে আমদানি করে এবং তারপর আনুষঙ্গিক কিছু কাজ করে ডিবাগিং শুরু করতে পারি।

২. পাইথন IDLE (আইডিএলই)-এ ডিবাগিং করতে পারি।

৩. টার্মিনালে বা কমান্ডলাইনে ডিবাগিং করতে পারি।

শেষেরটাই সবচেয়ে স্মার্ট সলিউশন। তাই আমরা এখন এটাই শিখব। তবে তার আগে test.py ফাইলে লিখে নেব একটা ছোট্ট প্রোগ্রাম, শুধু ডিবাগিংয়ের জন্য।

def square(x):

temp = x\*\*2

print(temp)

return

def main():

for i in range(1, 11):

square(i)

if \_\_name\_\_ == "\_\_main\_\_":

main()

আমরা এবার আমাদের টার্মিনালে কমান্ড দেব -

python3 -m pdb test.py

এই কমান্ডটি দিলেই টার্মিনালে ডিবাগার চালু হয়ে যাবে। অবশ্য এত বড় কমান্ড না দিয়ে আমরা চাইলে আরও ছোট একটা কমান্ডও দিতে পারি -

pdb3 test.py

ডিবাগার চালু হলে (Pdb) লেখার ডানপাশে বিভিন্ন কমান্ড দিয়ে আমরা ডিবাগিং করতে পারি।

এখন আমরা দরকারি কিছু ডিবাগিং কমান্ড শিখব:

| **কমান্ড** | **কাজ** |
| --- | --- |
| a বা args | বর্তমান ফাংশনের সবগুলো আর্গুমেন্ট ও তাদের ভ্যালু প্রিন্ট করে। |
| b বা break lineno | lineno-এর জায়গায় যে লাইন নম্বর দেওয়া হবে সেখানে এসে এক্সিকিউশন পজ হবে। |
| c বা continue | এক্সিকিউশন চলবে যতক্ষণ না পরবর্তী ব্রেকপয়েন্ট পাওয়া যায়। |
| n বা next | যতক্ষণ অবধি বর্তমান ফাংশনের পরবর্তী লাইন না পাওয়া যায় ততক্ষণ এক্সিকিউট হতে থাকে। |
| p expression | expression এক্সিকিউট হওয়ার পর এর ভ্যালু প্রিন্ট করে। |
| q বা quit | ডিবাগার বন্ধ করার জন্য। |
| restart | ডিবাগার রিস্টার্ট হয়। |
| s বা step | বর্তমান লাইন এক্সিকিউট করে পরবর্তী অকেশনের (নতুন ফাংশন বা বর্তমান ফাংশনের পরবর্তী লাইন) ঠিক প্রথম লাইনে থামবে। |
| w বা where | এখন কোনো ও কত নম্বর লাইন এক্সিকিউট হচ্ছে তা দেখা যায়। |
| whatis এক্সপ্রেশন | এক্সপ্রেশনের টাইপ প্রিন্ট করে। |

আরও অনেক আছে। তবে আপাতত এগুলো জানলেই কাজ চলে যাবে আমাদের। আরও বেশি জানার জন্য অবশ্যই অফিশিয়াল ডকের বিকল্প নেই।

> /home/maateen/Desktop/test.py(1)<module>()

-> def square(x):

(Pdb) s

> /home/maateen/Desktop/test.py(6)<module>()

-> def main():

(Pdb) b 6

Breakpoint 1 at /home/maateen/Desktop/test.py:6

(Pdb) s

> /home/maateen/Desktop/test.py(10)<module>()

-> if\_\_name\_\_=="\_\_main\_\_":

(Pdb) n

> /home/maateen/Desktop/test.py(11)<module>()

-> main()

(Pdb) c

1

4

9

16

25

36

49

64

81

100

The program finished and will be restarted

> /home/maateen/Desktop/test.py(1)<module>()

-> def square(x):

(Pdb) q

### লগিং

একটা প্রোগ্রাম যখন চলে, তখন অনেকগুলো ইভেন্ট ঘটে। এই ইভেন্টগুলো ট্রাক করে রেকর্ড রাখাই হলো লগিং। প্রোগ্রাম চলার সময় এটা বিভিন্ন ধরনের কাজ করে, এতে নানা ধরনের এরর ঘটতে পারে যা এক্সেপশনের মাধ্যমে হ্যান্ডেল করা হয়, এসব ঘটনা মেসেজ আকারে লগে লিখে রাখা হয়। যাতে প্রোগ্রাম ডিবাগ করতে সুবিধা হয়।

পাইথনের স্ট্যান্ডার্ড লাইব্রেরিতে খুবই শক্তিশালী একটা লগিং (logging) মডিউল আছে। সেটি ব্যবহার করে আমরা এখন একটি সিম্পল লগার (logger) তৈরি করব।

এ জন্য আমরা আমাদের test.py-তে কিছু কোড লিখব :

import logging

logging.basicConfig(filename='test.log', level=logging.INFO)

logging.debug('This is a debug message.')

logging.info('This is an informational message.')

logging.error('This is an error message.')

তৈরি করে ফেললাম একটা সিম্পল লগার। কিন্তু ঘটনা কি ঘটালাম আমরা? ব্যাখ্যা করা যাক।

logging.basicConfig(\*\*kwargs) ফাংশনটা ডিফল্ট ফরম্যাটারের সাথে একটা স্ট্রিমহ্যান্ডলার অবজেক্ট তৈরি করে। তারপর সেটাকে রুট লগারে অ্যাড করে আমাদের লগিং সিস্টেমের বেসিক কনফিগারেশন তৈরি করে। মোদ্দা কথা, আমাদের লগিং সিস্টেমের ইনিশিয়াল স্ট্রাকচার তৈরি করে দেয় আর কি! যাহোক, ওপরের প্রোগ্রামটা চালালে আমরা দেখব test.py এর সাথে একই ফোল্ডারে test.log নামে আরও একটা ফাইল তৈরি হয়েছে। সেখানে info এবং error দুটি মেসেজ আছে। কিন্তু debug মেসেজটা নেই। কারণ আমরা আমাদের লগারের মেসেজ লেভেল INFO-তে সেট করেছি। তাই এটা শুধু info, warning, critical ও error মেসেজই সেভ করবে ফাইলে।

এটা চেষ্টা করা যাক :

import logging

logging.basicConfig(filename='test.log', level=logging.INFO)

a = 10

b = 0

try:

temp = a/b

print(temp)

except ZeroDivisionError as e:

logging.exception(e)

এবার আমরা একটা রিয়েল লাইফ ইস্যুতে লগার ব্যবহার করলাম। আমরা ইতিমধ্যে জানি যে কোনো সংখ্যাকে শূন্য দিয়ে ভাগ করতে গেলে পাইথন ZeroDivisionError এক্সেপশন থ্রো করে। আমরা এই এক্সেপশনের পুরো এরর মেসেজটাই লগে সেভ করে রাখলাম logging.exception() ফাংশনের সাহায্যে। আরেকটা লক্ষ করার মতো ব্যাপার হলো, আমাদের আগের উদাহরণের সেই মেসেজ দুটিও কিন্তু রয়ে গেছে। কারণ হলো, ফাইলে সবকিছু অ্যাপেন্ড হচ্ছে। আমরা চাইলে ফাইলটাকে রাইট মোডেও ব্যবহার করতে পারি। এ জন্য logging.basicConfig(\*\*kwargs) এর মধ্যে filemode='w' সেট করতে হবে।

আমরা এখন আরেকটা উদাহরণ দেখব। তবে তার আগে আমাদের সেই hijibiji.py-তে আমরা যোগ করার ও কোনো ইন্টিজার জোড় সংখ্যা কি না, সেটা টেস্ট করার ফাংশন লিখব:

def add(a, b):

return a + b

def is\_even(number):

if (number % 2) == 0:

return True

else:

return False

কোড লেখা হয়ে গেছে। এবার তাহলে এর মধ্যে লগিং পুরে দেওয়া যাক। এ জন্য আমরা test.py-তে কোড লিখব :

import logging

from hijibiji import add, is\_even

logging.basicConfig(filename='test.log', level=logging.INFO)

logging.info('We are calling our add function.')

temp = add(12, 78)

print(temp)

logging.info('add function executed, task completed.')

logging.info('We are calling our is\_even funcion.')

temp = is\_even(2)

print(temp)

logging.info('is\_even function executed, task completed.')

প্রোগ্রামটা চালালে দেখব ফাইলে আরও কিছু ইনফরমেশনাল মেসেজ যোগ হয়েছে। কিন্তু মেসেজগুলো রসকষহীন। এগুলাকে এবার একটু স্পাইচি (সুস্বাদু) করা যাক। এ জন্য আমরা test.py-তে কোড লিখব:

import logging

from hijibiji import add, is\_even

logging.basicConfig(filename='test.log', format='%(asctime)s - %(name)s - %(levelname)s - %(message)s', level=logging.INFO)

logging.info('We are calling our add function.')

temp = add(12, 78)

print(temp)

logging.info('add function executed, task completed.')

logging.info('We are calling our is\_even funcion.')

temp = is\_even(2)

print(temp)

logging.info('is\_even function executed, task completed.')

এটা চালানোর পর লগফাইল চেক করলে দেখব, আগের চেয়ে মসলাদার হয়েছে মেসেজগুলো। লগিং নিয়ে আরও বেশি জানার জন্য অফিশিয়াল ডকুমেন্টেশন- <https://docs.python.org/3/library/logging.html->এ ঘুরে আসতে পারেন।

## টাইপ হিন্টিং

টাইপ হিন্টিং পাইথনের একেবারেই নতুন একটি ধারণা। এটা অনেকটা ভ্যারিয়েবল বা ফাংশন আর্গুমেন্টের টাইপ ডিক্লেয়ার করে দেয়ার মত। পাইথনে কোন ভ্যারিয়েবলের ডেটা টাইপ নির্ধারণ করে দেয়া লাগে না। ভ্যালু অ্যাসাইন করলে ভ্যারিয়েবলের ডেটা টাইপ আপনা-আপনি সেট হয়ে যায় - এটা আমরা ইতিমধ্যেই জেনেছি।

তাহলে আমাদের শিশুসুলভ মনে এ প্রশ্ন উঠতেই পারে, টাইপ হিন্টিং আবার কি জিনিস! একটা সংজ্ঞা তো আমরা উপরেই জেনেছি। কিন্তু সংজ্ঞায় কি আর প্রোগ্রামারদের মন ভরে? তো হয়ে যাক একটা উদাহরণ।

def add(a: int, b: int) -> int:

c = a + b

return c

উপরের ফাংশনটা একটু কিম্ভুতকিমাকার লাগছে, তাই না? আসলে এতক্ষণ আমরা যত ইউজার-ডিফাইন্ড ফাংশন দেখে এসেছি, তার কোনটার সুরতই এরকম নয়। সুরত ভিন্ন হবার কারণও রয়েছে। এই ফাংশনে আমরা টাইপ হিন্টিং ব্যবহার করেছি। টাইপ হিন্টিং করে বলে দিয়েছি a ও b আর্গুমেন্ট দুটি ইন্টিজার টাইপের হবে। আবার এটাও বলে দিয়েছি যে, add() ফাংশনটা যা রিটার্ন করবে তাও ইন্টিজার টাইপের হবে। যাহোক, এবার আরেকটা উদাহরণ দেখা যাক।

def add(a: int, b: int) -> int:

print(a, type(a))

print(b, type(b))

c = a + b

print(c, type(c))

return c

add(2, 3)

**আউটপুট**

2 <class 'int'>

3 <class 'int'>

5 <class 'int'>

এই উদাহরণের add() ফাংশনে আমরা দুটি ইন্টিজার টাইপের ভ্যালু আর্গুমেন্ট হিসেবে পাস করায় ফাংশনটি একটি ইন্টিজার টাইপের ভ্যালু রিটার্ন করেছে। একেবারে টাইপ হিন্টিংয়ে যেমনটা বলা ছিল, ঘটনা তেমনই ঘটেছে। ঠিক আছে! এবার আরো একটা উদাহরণ দেখা যাক।

def add(a: int, b: int) -> int:

print(a, type(a))

print(b, type(b))

c = a + b

print(c, type(c))

return c

add('Bangla', 'desh')

**আউটপুট**

Bangla <class 'str'>

desh <class 'str'>

Bangladesh <class 'str'>

এবার ঘটনা একটু রহস্যজনক। আর্গুমেন্ট হিসেবে ফাংশনের নেয়ার কথা ছিল ইন্টিজার টাইপের ভ্যালু অথচ আমরা পাস করেছি স্ট্রিং টাইপের ভ্যালু। আবার ফাংশনের রিটার্ন করার কথা ছিল ইন্টিজার টাইপের ভ্যালু অথচ রিটার্ন করেছে স্ট্রিং টাইপের ভ্যালু। আর সবচেয়ে আশংকাজনক কথা হল, পাইথন কোন এরর থ্রো করেনি। এই রহস্যের সমাধান কি করা যাবে? চেষ্টা করা যাক!

টাইপ হিন্টিং করলে আসলে রানটাইমে টাইপ চেক হয় না। এটা আসলে অ্যানোটেশন। আবার ডকুমেন্টেশনও বলতে পারি। ডকুমেন্টেশন এই অর্থে বললাম যে, add() ফাংশনের প্রথম লাইনের দিকে তাকিয়েই আমরা বলতে পারি a ও b ভ্যারিয়েবলের প্রত্যাশিত টাইপ হচ্ছে ইন্টিজার এবং ফাংশনটার প্রত্যাশিত রিটার্ন টাইপও ইন্টিজার। ব্যাপারটা হয়ত এখন আমাদের অনেকের কাছেই খুব একটা দরকারি মনে হচ্ছে না। কিন্তু যখন অন্য কোন ব্যক্তি আমাদের প্রোগ্রাম পড়বেন, তিনি কিন্তু খুব সহজেই বুঝতে পারবেন কোথায় কি ব্যবহার করে কি ফল পাওয়া যাবে।

আচ্ছা, পাইথনের জন্য কোন স্টাটিক টাইপ চেকার কি আসলেই নেই? এই প্রশ্নের উত্তরে না বললে মাইপাই প্রজেক্টের বদনাম করা হয়ে যাবে। মাইপাই হল পাইথনের (ঐচ্ছিক) স্টাটিক টাইপ চেকার। তবে এটা বিল্ট-ইন নয়। আমাদেরকে আলাদাভাবে ইন্সটল করে নিতে হবে। আর হ্যাঁ, এটা কিন্তু এখনও ডেভেলপমেন্ট ফেইজে রয়েছে।

sudo pip3 install mypy

এবার মাইপাই দিয়ে একটু আগে লেখা প্রোগ্রামটা চেক করা যাক। ধরে নিচ্ছি, প্রোগ্রামটা আমরা type\_hint.py ফাইলে লিখেছি।

mypy type\_hint.py

**আউটপুট**

type\_hint.py:8: error: Argument 1 to "add" has incompatible type "str"; expected "int"

type\_hint.py:8: error: Argument 2 to "add" has incompatible type "str"; expected "int"

মাইপাই আমাদের চমৎকারভাবে বলে দিচ্ছে যে, আমাদের প্রোগ্রামের আট নম্বর লাইনের add() ফাংশনে ঘাপলা আছে। আর‌ো বলে দিচ্ছে যে, আমরা a ও b আর্গুমেন্টে ইন্টিজার টাইপের বদলে স্ট্রিং টাইপের ভ্যালু পাস করেছি।

### ভ্যারিয়েবল অ্যানোটেশন

টাইপ হিন্টিং সম্পর্কে ধারণা নিতে গিয়ে আমরা এক দৌড়ে একেবারে সাগরে গিয়ে পড়েছি। তাই এবার একটু পিছনে ফিরে তাকাব আর ধাপে ধাপে সাঁতার শেখার চেষ্টা করব। প্রথমেই ভ্যারিয়েবল অ্যানোটেশন দিয়ে শুরু করা যাক।

number: int = 10

value = 3.1415 # type: float

name: str = 'maateen'

address = 'Dhaka' # type: str

is\_happy: bool = True

is\_sad = False #type: bool

উপরের উদাহরণ থেকে আমরা বুঝতে পারছি যে, ভ্যারিয়েবল অ্যানোটেশন দুইভাবে করা সম্ভব। এই অধ্যায়ের শুরুর দিকে ফাংশনের আর্গুমেন্টকে যেভাবে অ্যানোটেট করেছি, তার পাশাপাশি কমেন্ট করেও ভ্যারিয়েবল অ্যানোটেশন সম্ভব। এই উদাহরণে আমরা ইন্টিজার, ফ্লোট, স্ট্রিং ও বুলিয়ান টাইপের ভ্যারিয়েবল অ্যানোটেট করেছি। এবার লিস্ট, টাপল ও ডিকশনারি টাইপের ভ্যারিয়েবল অ্যানোটেট করার চেষ্টা করা যাক।

numbers: tuple[int]

planets = ['Earth', 'Mars', 'Jupiter'] # type: list[str]

মাইপাই দিয়ে চেক করলে আমরা নিচের মত দুটি এরর দেখতে পাব।

type\_hint.py:1: error: "tuple" is not subscriptable, use "typing.Tuple" instead

type\_hint.py:2: error: "list" is not subscriptable, use "typing.List" instead

এই এররগুলো কি বলতে চাচ্ছে? উত্তরটা খুব সিম্পল – টাইপিং মডিউল ব্যবহার করতে বলছে। পাইথনের ৩.৫ সংস্করণ থেকে এই মডিউলটি স্টান্ডার্ড লাইব্রেরিতে প্রভিশনার বেসিসে যোগ করা হয়েছে। প্রভিশনাল বেসিস বলতে বুঝায় যার কোন ব্যাকওয়ার্ড কম্প্যাটিবিলিটি গ্যারান্টি নাই। যাহোক, আমরা আমাদের গল্পে ফিরে আসি।

from typing import Dict, List, Tuple

numbers: Tuple[int]

planets = ['Earth', 'Mars', 'Jupiter'] # type: List[str]

books: Dict[str, str]

টাইপিং মডিউল Dict, List, Tuple, Any, Union ইত্যাদি টাইপ হিন্টিং সাপোর্ট করে। Any মানে হল টাইপের কোন বাদ-বিচার নাই, স্ট্রিং-ও হতে পারে আবার ইন্টিজারও হতে পারে। List ও Tuple-এর ক্ষেত্রে তাতে কোন ধরনের ভ্যারিয়েবল থাকবে তাও বলে দিতে হয়। আর Dict-এর ক্ষেত্রে কী ও ভ্যালু - উভয়ের টাইপই বলে দিতে হয়।

### ক্লাস এবং ইন্সট্যান্স ভ্যারিয়েবল অ্যানোটেশন

ক্লাস এবং ইন্সট্যান্স ভ্যারিয়েবল অ্যানোটেশন একদম সিম্পল জিনিস। একটা উদাহরণ দেখা যাক।

from typing import ClassVar

class Human:

name: str

age: int

gender: str

address: ClassVar[str] = 'Dhaka'

def \_\_init\_\_(self, name: str = 'maateen') -> None:

self.name = name

এই উদাহরণে name, age ও gender হল ইন্সট্যান্স ভ্যারিয়েবল আর address হল ক্লাস ভ্যারিয়েবল। ক্লাস ভ্যারিয়েবলকে টাইপিং মডিউলের ClassVar ক্লাস দিয়ে অ্যানোটেট করেছি আমরা। আরেকটা কথা, ClassVar ব্যবহার করার সময় এর প্যারামিটার হিসেবে কোন টাইপ ভ্যারিয়েবল ব্যবহার করা যাবে না।

### টাইপ অ্যালিয়াস

অ্যালিয়াস (alias)-এর বাংলা হল উপনাম বা ওরফে। আমরা অনেক সময় টিভিতে খবরে এরকমটা শুনে থাকি - র‍্যাবের সাথে বন্দুকযুদ্ধে কুখ্যাত সন্ত্রাসী মোহাম্মদ হুমায়ুন কবির ওরফে গালকাটা কবির নিহত। এখানে, মোহাম্মদ হুমায়ুন কবির উক্ত লোকের আসল নাম আর গালকাটা কবির হল তার উপনাম বা অ্যালিয়াস।

টাইপ অ্যালিয়াস হল কোন একটা টাইপকে অন্য নামে ডাকা অথবা আরো ভালভাবে বলতে গেলে, অন্য ভ্যারিয়েবলে অ্যাসাইন করা।

from typing import Dict, List

HostName = str

Address = str

Server = Dict[HostName, Address]

Network = List[Server]

উপরের উদাহরণটিতে HostName ও Address হল স্ট্রিংয়ের (str) টাইপ অ্যালিয়াস এবং Server হল ডিকশনারির (Dict) টাইপ অ্যালিয়াস যা আবার HostName ও Address এই দুটি টাইপের সমন্বয়ে গঠিত। সবশেষে, Network হল লিস্টের (List) টাইপ অ্যালিয়াস যাতে প্রত্যেকটি আইটেম আবার Server টাইপের।

### নিউ টাইপ

টাইপকে অন্য নামে ডাকাডাকি অনেক তো হল। এবার সময় নতুন টাইপ তৈরি করার। আর এজন্য আমাদেরকে সাহায্য করবে NewType() হেল্পার ফাংশন। এই ফাংশনটি দুটি প্যারামিটার নেয় - নতুন টাইপের নাম (name) এবং টাইপ (tp)। একটা উদাহরণ দেখা যাক।

from typing import Dict, List, NewType

HostName = NewType('HostName', str)

Address = NewType('Address', str)

Server = NewType('Server', Dict[HostName, Address])

Network = NewType('Network', List[Server])

টাইপ অ্যালিয়াসের উদাহরণটাই আমরা আবার নতুন মোড়কে দেখছি। এখানে আমরা HostName, Address, Server ও Network নামের নতুন কিছু টাইপ তৈরি করেছি। আচ্ছা, তৈরি তো করলাম। এবার ব্যবহার করা যাক।

hostname = HostName('local')

address = Address('127.0.0.1')

server = Server({hostname: address})

network = Network([server])

পূর্বের উদাহরণগুলোর তুলনায় একেবারেই ভিন্ন, তাই না? আচ্ছা, আগের মত করে টাইপগুলোকে ব্যবহার করলে কি সমস্যা হত? সেই তদন্তের দায়ভার শুধুই আপনাদের। ওহ! আরেকটা কথা, এই তদন্ত শেষ না করে সামনে আগানোতে শার্লক হোমসের নিষেধ আছে।

### টাইপ ভ্যারিয়েবল

টাইপ ভ্যারিয়েবল সম্পর্কে অল্প-বিস্তর জানার পূর্বে আমরা একটা উদাহরণ দেখব।

from typing import TypeVar

A = TypeVar('A')

B = TypeVar('B', str)

C = TypeVar('C', str, int)

def add(x: A, y: C) -> A:

pass

**আউটপুট**

type\_hint.py:4: error: TypeVar cannot have only a single constraint

type\_hint.py:4: error: "object" not callable

আমরা ইতিমধ্যে str, int, bool প্রভৃতি টাইপের সাথে পরিচিত হয়েছি এবং কোন ভ্যারিয়েবলের সামনে কোলন চিহ্ন দিয়ে এদেরকে ব্যবহারও করেছি। এদেরকে জেনেরিক টাইপ বলা হয়। আর NewType() ফাংশন ব্যবহার করে আমরা যে নতুন টাইপ তৈরি করেছিলাম, তা কিন্তু জেনেরিক টাইপ ছিল না। এইজন্যই ভ্যারিয়েবলের সাথে কোলন চিহ্ন দিয়ে আমরা সেগুলোকে ব্যবহার করতে পারিনি।

এবার আমাদের উদাহরণে ফেরা যাক। TypeVar() ফাংশন দিয়ে আমরা নতুন জেনেরিক টাইপ তৈরি করতে পারি। আউটপুটে দেখতে পাচ্ছি চার নাম্বার লাইনে অর্থাৎ B = TypeVar('B', str) লাইনে দুটো এরর থ্রো করেছে মাইপাই। কারণটা হল, TypeVar() ফাংশনটি নতুন টাইপের নামের পাশাপাশি কমপক্ষে দুটি টাইপকে প্যারামিটার হিসেবে নিয়ে থাকে। আর যেহেতু আমরা চার নাম্বার লাইনে শুধু একটি (str) টাইপ পাস করেছি, তাই মাইপাই এরর থ্রো করেছে।

তবে এসবের পরেও ঝামেলা কিন্তু একটু রয়েছে। আর সেটা হল এই নতুন টাইপকে ব্যবহার করার ক্ষেত্রে। কিন্তু বইয়ের এই পর্যায়ে এসে এরকম আট-দশটা সমস্যার সমাধান তো আপনাদের থেকে আশা করাই যেতে পারে।

## স্টাইল গাইড

প্রোগ্রামারদের কত বেশি স্বাধীনতা প্রয়োজন তার একটি এক্সপেরিমেন্ট হচ্ছে পাইথন। অতিরিক্ত স্বাধীনতার ফলে যেমন একজনের লেখা কোড অন্যজন পড়তে পারে না; ঠিক তেমনি স্বল্প স্বাধীনতা কোডের প্রকাশ ভঙ্গিকে বাধাগ্রস্ত করে। - গুইডো ভ্যান রস্যিউম

আমরা এখন PEP-8 স্টাইল গাইড সম্পর্কে জানব। এটি হলো পাইথনের স্টাইল গাইড। মানে কীভাবে একটা ভেরিয়েবলের নাম ডিক্লেয়ার করা উচিত, কীভাবে ফাংশনের বা ক্লাসের নাম লেখা উচিত, কীভাবে ইনডেন্টেশন করা উচিত ইত্যাদির নীতিমালা।

কোডিং কনভেনশনের মূল কারণ হলো কোডকে হাইলি রিড্যাবল (অত্যন্ত সুখপাঠ্য) করা। আমরা যা লিখলাম অন্য কেউ যদি তা না-ই বুঝতে পারে তাহলে লিখে কী লাভ। এই ভাবনা থেকেই স্টাইল গাইডের উৎপত্তি।

অনেক প্রজেক্টের (যেমন: জ্যাঙ্গো) নিজস্ব স্টাইল গাইড আছে। এক্ষেত্রে PEP-8 স্টাইল গাইডের পরিবর্তে সেসব স্টাইল গাইড মানাই শ্রেয়। যাহোক, PEP-8 স্টাইল গাইডের নিয়মগুলো আলোচনা করা যাক।

### কোড লে-আউট

#### ইনডেন্টেশন

কোড ইনডেন্ট করার সময় আমরা সব সময় চারটি স্পেস ব্যবহার করব। আর লাইন কন্টিনিউ (continue) করার সময় প্রথম, দ্বিতীয় বা তৃতীয় বন্ধনীর ক্ষেত্রে এদের সাথে ভার্টিকালি (উল্লম্বভাবে) ইনডেন্ট করব।

foo = long\_function\_name(var\_one, var\_two,

var\_three, var\_four)

ভার্টিকালি ইনডেন্ট না করলে প্রথম লাইনে আর্গুমেন্ট লেখা দূষণীয়।

foo = long\_function\_name(var\_one, var\_two,

var\_three, var\_four)

আমরা চাইলে ভার্টিকালি ইন্ডেন্ট না করে হ্যাংগিং (hanging) ইন্ডেন্টও ব্যবহার করতে পারি। এইক্ষেত্রে প্রথম লাইনে কোন আর্গুমেন্ট রাখব না আর পরের লাইন থেকে লাইন কন্টিনিউয়েশনের (continuation) মত করে ইন্ডেন্ট করব।

foo = long\_function\_name(

var\_one, var\_two,

var\_three, var\_four)

if (this\_is\_one\_thing and

that\_is\_another\_thing):

do\_something()

কমেন্ট যোগ করলেও চারটা স্পেস দিয়ে ইন্ডেন্ট করব আমরা।

if (this\_is\_one\_thing and

that\_is\_another\_thing):

# Since both conditions are true, we can frobnicate.

do\_something()

তবে ফাংশনের আর্গুমেন্ট বেশি হলে চারটার বদলে আটটা স্পেস দিয়ে শুধু আর্গুমেন্টগুলোকে ইন্ডেন্ট করলে ভাল হয়।

def long\_function\_name(

var\_one, var\_two, var\_three,

var\_four):

print(var\_one)

তবে এরকমটা করা দূষণীয়:

def long\_function\_name(

var\_one, var\_two, var\_three,

var\_four):

print(var\_one)

কন্ডিশনাল লজিকের কন্ডিশন বেশি লম্বা হলে লাইন কন্টিনিউয়েশন হিসাবে ইন্ডেন্ট করা যায়।

if (this\_is\_one\_thing

and that\_is\_another\_thing):

do\_something()

প্রথম, দ্বিতীয় ও তৃতীয় বন্ধনীর জোড়ের ক্ষেত্রে শেষ বন্ধনীটা কয়েকভাবে ইনডেন্ট করা যায়।

my\_list = [

1, 2, 3,

4, 5, 6,

]

result = some\_function\_that\_takes\_arguments(

'a', 'b', 'c',

'd', 'e', 'f',

)

my\_list = [

1, 2, 3,

4, 5, 6,

]

result = some\_function\_that\_takes\_arguments(

'a', 'b', 'c',

'd', 'e', 'f',

)

#### ট্যাব নাকি স্পেস

আমরা সবসময় স্পেস দিয়েই ইনডেন্ট করব। তবে কোন প্রোগ্রামে যদি ইতিমধ্যে ট্যাব ব্যবহার করা হয়ে থাকে, সেখানে ট্যাব ব্যবহার করব আমরা। কারণ পাইথন-৩ এ স্পেস ও ট্যাবের মিশ্রণ দূষনীয়।

#### লাইন লেন্থ

একটা লাইন সর্বোচ্চ ৭৯ ক্যারেক্টারের হতে পারবে। তবে ডকস্ট্রিং বা কমেন্টের মত বড়সড় টেক্সট ব্লকের ক্ষেত্রে প্রতিটা লাইন ৭২ ক্যারেক্টারের ভিতর সীমবদ্ধ থাকা উচিত। সীমার চেয়ে বড় হয়ে গেলে লাইন কন্টিনিউয়েশন ইন্ডেন্টেশন ব্যবহার করব। যেসব ক্ষেত্রে সাধারণ কন্টিনিউয়েশন ইন্ডেন্টেশন ব্যবহার করা যায় না সেখানে ব্যাকস্লাশ \ ব্যবহার করব।

with open('/path/to/some/file/you/want/to/read') as file\_1, \

open('/path/to/some/file/being/written', 'w') as file\_2:

file\_2.write(file\_1.read())

#### বাইনারি অপারেটর আর লাইন ব্রেক

বহু বছর ধরে, বাইনারি অপারেটর ব্যবহারের ক্ষেত্রে লাইন ব্রেক অপারেটরের পরে হচ্ছে।

income = (gross\_wages +

taxable\_interest +

(dividends - qualified\_dividends) -

ira\_deduction -

student\_loan\_interest)

কিন্তু তা রিড্যাবিলিটি নষ্ট করে বলে আমরা সবসময় বাইনারি অপারেটর ব্যবহারের ক্ষেত্রে লাইন ব্রেক অপারেটরের আগে ব্যবহার করব।

income = (gross\_wages

+ taxable\_interest

+ (dividends - qualified\_dividends)

- ira\_deduction

- student\_loan\_interest)

#### ব্লাঙ্ক লাইন - শূন্যে ভরা জীবন

টপ লেভেল ফাংশন ও ক্লাস ডেফিনিশনের আগে-পরে দুইটা ব্লাঙ্ক লাইন দিব আমরা।

ক্লাসের ভিতর মেথড ডেফিনিশনের আগে-পরে একটা ব্লাঙ্ক লাইন দিব আমরা।

#### ইমপোর্ট - আমদানি প্রকল্প

আমদানির ক্ষেত্রে সবসময় সেপারেট লাইন ব্যবহার করব আমরা।

import os

import sys

নিচের মত কখনোই করব না।

import sys, os

তবে from … import এর হিসাব আলাদা।

from subprocess import Popen, PIPE

মডিউল আমদানি সবসময় ফাইলের শুরুতে হবে। তবে একটা মডিউল আমদানির আগে সেই সম্পর্কিত কমেন্ট বা ডকস্ট্রিং থাকতে পারে। আর মডিউলগুলি বিভিন্ন গ্রুপে ভাগ করে আমদানি করব আমরা। প্রথমে আমদানি করব স্টান্ডার্ড মডিউল, তারপর করব থার্ড-পার্টি মডিউল এবং সবার শেষে লোকাল মডিউল। প্রতিটা গ্রুপকে ব্লাঙ্ক লাইন দিয়ে আলাদা করব আমরা। আর পারতপক্ষে ওয়াইল্ডকার্ড আমদানি (from <module> import \*) করব না।

হাইলি রিড্যাবল বলে যথাসম্ভব অ্যাবসলিউট ইমপোর্ট ব্যবহার করব।

import mypkg.sibling

from mypkg import sibling

from mypkg.sibling import example

কমপ্লেক্স প্যাকেজের ক্ষেত্রে রিলেটিভ ইমপোর্ট ব্যবহার করা যায়।

from . import sibling

from .sibling import example

কোন মডিউল থেকে কোন ক্লাসকে আমদানি করতে চাইলে এভাবে করব:

from myclass import MyClass

from foo.bar.yourclass import YourClass

অবশ্য লোকাল ক্লাসের সাথে ক্লাশ হলে এভাবে আমদানি করব:

import myclass

import foo.bar.yourclass

আর ব্যবহার করার ক্ষেত্রে myclass.MyClass ও foo.bar.yourclass.YourClass এভাবে ব্যবহার করব।

#### মডিউল লেভেল ডান্ডার নেইম

মডিউল লেভেল ডান্ডার নেইম (যেমন: \_\_all\_\_, \_\_author\_\_, \_\_version\_\_) সবসময় মডিউল ডকস্ট্রিংয়ের পরে আর কোন মডিউল আমদানি করার আগে (from \_\_future\_\_ import ... ব্যতীত) ডিক্লেয়ার করব।

"""This is the example module.

This module does stuff.

"""

from \_\_future\_\_ import barry\_as\_FLUFL

\_\_all\_\_ = ['a', 'b', 'c']

\_\_version\_\_ = '0.1'

\_\_author\_\_ = 'Cardinal Biggles'

import os

import sys

### স্ট্রিং এবং ডকস্ট্রিং

আমরা আগেই জেনেছি, পাইথনে সিঙ্গেল-কোটেড বা ডবল-কোটেড স্ট্রিং মোটামুটি একই জিনিস। PEP-8 এ এদের নিয়ে কোন রিকমেন্ডেশন নাই। তবে ট্রিপল-কোটেড স্ট্রিং বা ডকস্ট্রিং নিয়ে PEP-257 এ কিছু নিয়ম-নীতি আছে।

#### ডকস্ট্রিং কী?

ডকস্ট্রিং হল এমন এক ধরনের স্ট্রিং যা মডিউল, ফাংশন বা মেথড ও ক্লাসে প্রথম স্টেটমেন্ট হিসাবে থাকে। এধরনের স্ট্রিং ঐ অবজেক্টের স্পেশাল অ্যাট্রিবিউট \_\_doc\_\_ হিসাবে বিবেচিত হয়। মোটামুটি সব মডিউল, ফাংশন বা মেথড ও ক্লাসে ডকস্ট্রিং থাকা উচিত। ডকস্ট্রিং phrase এর মত সংক্ষিপ্ত হবে, নির্দিষ্ট পিরিয়ড পর পর শেষ হবে। ফাংশন বা মেথডের ইফেক্ট কমান্ড হিসাবে থাকবে, বর্ণনা হিসাবে নয়। ডকস্ট্রিংয়ের দুইটা ফর্ম আছে - ওয়ান-লাইন ডকস্ট্রিং ও মাল্টি-লাইন ডকস্ট্রিং।

#### ওয়ান-লাইন ডকস্ট্রিং

নাম শুনেই বোঝা যাচ্ছে, এসব ডকস্ট্রিং থাকবে মাত্র এক লাইন জুড়ে এবং সর্বোচ্চ ৭৯ ক্যারেক্টারের হবে লাইনটি। এর আগে-পিছে কোন ব্লাঙ্ক লাইন থাকবে না। তবে ক্লাসের ডকস্ট্রিংয়ের পর একটা ব্লাঙ্ক লাইন রাখা উচিত।

def kos\_root():

"""Return the pathname of the KOS root directory."""

global \_kos\_root

if \_kos\_root: return \_kos\_root

...

#### মাল্টি-লাইন ডকস্ট্রিং

ডকস্ট্রিং কয়েকটা লাইনে লেখা হয়। সাধারণত ক্লাসের ক্ষেত্রে মাল্টি-লাইন ডকস্ট্রিং ব্যবহৃত হয়। প্রথমে ক্লাসের বর্ণনা আর তারপরে আর্গুমেন্টের বর্ণনা থাকে। আর্গুমেন্টের বর্ণনার ক্ষেত্রে প্রতিটি আর্গুমেন্টের জন্য পৃথক লাইন বরাদ্দ করা উচিত।

def complex(real=0.0, imag=0.0):

"""Form a complex number.

Keyword arguments:

real -- the real part (default 0.0)

imag -- the imaginary part (default 0.0)

"""

if imag == 0.0 and real == 0.0:

return complex\_zero

...

#### হোয়াইটস্পেস

প্রথম, দ্বিতীয় ও তৃতীয় বন্ধনীর সাথে স্পেস দেয়া যাবে না।

spam(ham[1], {eggs: 2})

নিচের মত হবে না।

spam( ham[ 1 ], { eggs: 2 } )

কমা, সেমিকোলন ও কোলনের আগে স্পেস দেয়া যাবে না।

if x == 4: print x, y; x, y = y, x

নিচের মত হবে না।

if x == 4 : print x , y ; x , y = y , x

বাইনারি অপারেটরের আগে-পরে স্পেস না দিলেও চলে। তবে দিলে সমান সংখ্যক স্পেস হবে। কোলন যখন বাইনারি অপারেটর তখন কোলনের ক্ষেত্রেও এই নিয়ম প্রযোজ্য।

ham[1:9], ham[1:9:3], ham[:9:3], ham[1::3], ham[1:9:]

ham[lower:upper], ham[lower:upper:], ham[lower::step]

ham[lower+offset : upper+offset]

ham[: upper\_fn(x) : step\_fn(x)], ham[:: step\_fn(x)]

ham[lower + offset : upper + offset]

নিচের মত হবে না।

ham[lower + offset:upper + offset]

ham[1: 9], ham[1 :9], ham[1:9 :3]

ham[lower : : upper]

ham[ : upper]

ফাংশনের আর্গুমেন্ট বা ইনডেক্সিং শুরুর বন্ধনীর আগে স্পেস হবে না।

spam(1)

dct['key'] = lst[index]

নিচের মত হবে না।

spam (1)

dct ['key'] = lst [index]

অ্যাসাইনমেন্ট অপারেটরের আগে পরে কেবল একটি করে স্পেস হবে।

x = 1

y = 2

long\_variable = 3

নিচের মত হবে না।

x = 1

y = 2

long\_variable = 3

যদি কয়েকটি অপারেটরের মধ্যে গুরুত্ব কম-বেশি থাকে, তবে যেটার গুরুত্ব সবচেয়ে কম, এর আগে-পরে স্পেস দিতে হবে। যেমন -

i = i + 1

submitted += 1

x = x\*2 - 1

hypot2 = x\*x + y\*y

c = (a+b) \* (a-b)

নিচের মত হবে না।

i=i+1

submitted +=1

x = x \* 2 - 1

hypot2 = x \* x + y \* y

c = (a + b) \* (a - b)

ফাংশনের কীওয়ার্ড আর্গুমেন্ট বা ডিফল্ট প্যারামিটারের ক্ষেত্রে = চিহ্নের আগে-পরে কোন স্পেস হবে না।

def complex(real, imag=0.0):

return magic(r=real, i=imag)

নিচের মত হবে না।

def complex(real, imag = 0.0):

return magic(r = real, i = imag)

### নেমিং কনভেনশন

পাইথনে নেমিং কনভেনশন একটু জগাখিঁচুড়ি টাইপের, আমরা কখনোই সামঞ্জস্যপূর্ণ কিছু পাব না এখানে। তাই আমরা আপাতত স্টান্ডার্ড কিছু কনভেনশনে চোখ বুলাব।

* b (একটামাত্র ছোট হাতের অক্ষর)
* B (একটামাত্র বড় হাতের অক্ষর)
* lowercase
* lower\_case\_with\_underscores
* UPPERCASE
* UPPER\_CASE\_WITH\_UNDERSCORES
* CapitalizedWords (অথবা CapWords বা CamelCase – উটের পিঠের মত বলে এরকম নামকরণ)। StudlyCaps নামেও পরিচিত।

নোট: CapWords এ অ্যাব্রিভিয়েশন ব্যবহার করার ক্ষেত্রে অ্যাব্রিভিয়েশনের সব কয়টা লেটারকে বড় হাতের অক্ষর করতে হবে। এইভাবে HTTPServerError, HttpServerError থেকে উত্তম নামকরণ।

* mixedCase (শুরুতে ছোট হাতের অক্ষর কিন্তু পরে বড় হাতের অক্ষর)
* Capitalized\_Words\_With\_Underscores (বিশ্রী নামকরণ!)

সি++ বা জাভার মত পাইথনে পাবলিক, প্রাইভেট আইডিয়া নাই। পাইথনে সবকিছুই পাবলিক। তবে কনভেনশন দ্বারা এদের ব্যবহার আলাদা করা যায়। ক্লাস, মেথডের নামের আগে একটা আন্ডারস্কোর থাকলে তাকে প্রোটেক্টেড ধরা যায়। অপরপক্ষে ডাবল আন্ডারস্কোর থাকলে প্রাইভেট ধরা হয়।

### পরিহারযোগ্য নাম

নামের ক্ষেত্রে কখনো l (এল), O (ও) এবং I (আই) ব্যবহার করা ঠিক না। বিভিন্ন ফন্টে এরা মিলেমিশে একাকার হয়ে যেতে পারে।

### প্যাকেজ ও মডিউলের নাম

মডিউলের নাম ছোট হবে, সবগুলা ছোট হাতের অক্ষর হবে। আন্ডারস্কোর ব্যবহার করা যায় যদি রিড্যাবিলিটির উন্নতি হয় তাইলে। আর প্যাকেজের নামও ছোট হবে, সবগুলা ছোট হাতের অক্ষর হবে। তবে আন্ডারস্কোর না থাকাই শ্রেয়।

### ক্লাস ও এক্সসেপশনের নাম

ক্লাসের নামের ক্ষেত্রে সাধারণত CapWords কনভেনশন মানা হয়। এক্সসেপশন যেহেতু ক্লাস তাই এর নামও CapWords কনভেনশন মেনে হবে। তবে নামের শেষে (নামের সাথেই) অবশ্যই Error কথাটা থাকতে হবে।

### গ্লোবাল ভ্যারিয়েবল ও ফাংশনের নাম

এদের নাম হবে ছোট হাতের অক্ষরের ও অর্থবহ। রিড্যাবিলিটি উন্নত করার জন্য ওয়ার্ডগুলোকে আন্ডারস্কোর দিয়ে আলাদা করা উচিত। যদি কোন ইউজার ডিফাইন্ড ফাংশনের নাম কোন রিজার্ভড কীওয়ার্ড বা বিল্ট-ইন ফাংশনের সাথে ক্লাশ করে তাহলে নামের শেষে একটা আন্ডারস্কোর দেয়াই উত্তম। কথাটি আর্গুমেন্টের ক্ষেত্রেও সত্য।

### প্রোগ্রামিং রিকমেন্ডেশন

* None এর মত সিঙ্গেলটনের সাথে কম্পেয়ার করার ক্ষেত্রে == অপারেটরের পরিবর্তে is অথবা is not ব্যবহার করা উচিত। অন্যদিকে not ... is ব্যবহার করার চেয়ে is not অপারেটর ব্যবহার করাই উত্তম।

if foo is not None:

নিচের মত হবে না।

if not foo is None:

* ল্যাম্বডাকে কোন আইডেন্টিফায়ারে অ্যাসাইন করে কাজ সিদ্ধ না করে সব সময় def স্টেটমেন্ট ব্যবহার করা উচিত।

def f(x): return 2\*x

নিচের মত হবে না।

f = lambda x: 2\*x

* কাস্টম এক্সেপশন পয়দা করার সময় BaseException থেকে ডিরাইভ না করে Exception থেকে ডিরাইভ করা উচিত।
* এক্সেপশন ক্যাচ করার সময় বেয়ার (bare) except: এর জন্য কোড না লিখে যথাসম্ভব স্পেসিফিক এক্সেপশনের জন্য কোড লেখা উচিত। বেয়ার (bare) except: SystemExit ও KeyboardInterrupt এক্সেপশন ক্যাচ করে ও অন্যান্য প্রব্লেমকে ডিজগাইজ(disguise) করে। বেয়ার (bare) except: হল except BaseException: এর সমতুল্য। যাহোক, যদি আমরা সবগুলো এক্সেপশনকে একসাথে ধরতে চাই আবার প্রব্লেমও রেইজ করতে চাই তাহলে except Exception: ব্যবহার করতে হবে।

try:

import platform\_specific\_module

except ImportError:

platform\_specific\_module = None

* সকল প্রকার try ... except এর ক্ষেত্রে try ব্লকে যথাসম্ভব কম কোড রাখা উচিত। এর ফলে বাগ-ভাল্লুক লুকিয়ে থাকার সুযোগ পাবে না।
* ফাংশনের রিটার্ন স্টেটমেন্টে সঙ্গতিপূর্ণ কিছু রিটার্ন করা উচিত। হয় কোন এক্সপ্রেশন রিটার্ন করা উচিত নয়তো None রিটার্ন করা উচিত। আর ফাংশনের শেষে অবশ্যই একটা রিটার্ন স্টেটমেন্ট থাকা উচিত।

def foo(x):

if x >= 0:

return math.sqrt(x)

else:

return None

def bar(x):

if x < 0:

return None

return math.sqrt(x)

নিচের মত হবে না।

def foo(x):

if x >= 0:

return math.sqrt(x)

def bar(x):

if x < 0:

return

return math.sqrt(x)

* লিস্ট, টাপল, ডিকশনারির এম্পটি কিনা সেটা চেক করার যায় নিচের মত করে:

if not seq:

if seq:

নিচের মত হবে না।

if len(seq):

if not len(seq):

* বুলিয়ান ভ্যালুকে == অপারেটর ব্যবহার করে True বা False এর সাথে তুলনা করা উচিত না।

if greeting:

নিচের মত হবে না।

if greeting == True:

if greeting is True:

এই ছিল মোটামুটি একটা সংকলন। আরো বিস্তারিত জানার জন্য আমাদেরকে অফিসিয়াল ডক ( <https://www.python.org/dev/peps/pep-0008/> ) ফলো করতে হবে।

## বেঞ্চমার্কিং

অধ্যায়ের নামটি দেখে আমাদের অনেকের ভ্রুই হয়তো দুই হাত কুঁচকে উঠেছে। বেঞ্চমার্কিং? এটা আবার কোন চিজ?

উইকিপিডিয়া বলে-

কতগুলো স্ট্যান্ডার্ড টেস্ট ও ট্রায়াল চালিয়ে কোনো অবজেক্টের রিলেটিভ পারফরম্যান্স (আপেক্ষিক কর্মক্ষমতা) মূল্যায়ন করতে কোনো কম্পিউটার প্রোগ্রাম বা কয়েকটা প্রোগ্রাম অথবা অন্য কোনো অপারেশন চালানোকেই বেঞ্চমার্কিং বলে।

আরও সহজভাবে বলা যায়, একটা প্রোগ্রাম কতটুকু দ্রুত চলে বা কোথায় এর চলার গতি ব্যাহত হয়, তা খুঁজে বের করার প্রক্রিয়াই হলো বেঞ্চমার্কিং। সাধারণত চারটি প্রশ্নের উত্তর খুঁজে পেতে একটা প্রোগ্রামের বেঞ্চমার্কিং করা হয়-

* এটি কত দ্রুত চলে?
* কোথায় এর গতি ব্যাহত হচ্ছে?
* কতটুকু মেমোরি এটি ব্যবহার করছে?
* কোথায় মেমোরি লিকিং হচ্ছে অথবা কোন ব্লকটা বেশি মেমোরি খাচ্ছে (ব্যবহার করছে)?

আমরা এখন এই চার প্রশ্নের উত্তর খুঁজে বের করতে কিছু টুলসের সহায়তা নেব। তবে তার আগে test.py-তে আমরা নিচের মতো করে একটা প্রোগ্রাম লিখে নেব। এই প্রোগ্রামটার ওপরেই আমরা কারুকাজ চালাব।

### time

লিনাক্সের মান্ধাতার আমলের টাইম টুল দিয়েই শুরু করা যাক। সম্ভবত এটি ‘ম্যাকওএস’-এও আছে। এই টুল ব্যবহার করে একটি প্রোগ্রাম চলতে কোন খাতে কতটুকু সময় ব্যয় করে তার হিসাব পাওয়া যায়। আমাদের test.py-এর বেঞ্চমার্কিং করতে নিচের মতো করে টার্মিনালে কমান্ড দেব।

time -v python3 test.py

এ ক্ষেত্রে আমরা নিচের মতো কিছু আউটপুট পাব—

100000 loops.

Command being timed: "python3 test.py"

User time (seconds): 0.05

System time (seconds): 0.00

Percent of CPU this job got: 100%

Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.05

Average shared text size (kbytes): 0

Average unshared data size (kbytes): 0

Average stack size (kbytes): 0

Average total size (kbytes): 0

Maximum resident set size (kbytes): 9392

Average resident set size (kbytes): 0

Major (requiring I/O) page faults: 0

Minor (reclaiming a frame) page faults: 1105

Voluntary context switches: 0

Involuntary context switches: 8

Swaps: 0

File system inputs: 0

File system outputs: 0

Socket messages sent: 0

Socket messages received: 0

Signals delivered: 0

Page size (bytes): 4096

Exit status: 0

সবটুকু আমাদের মাথাব্যথার কারণ নয়। কেবল User time, System time ও Elapsed time নিয়ে আমরা চিন্তা করব। এই তিন ধরনের সময়ের ভিন্ন ভিন্ন অর্থ আছে।

* ইউজার টাইম (User time): কার্নেলের বাইরে খরচ হওয়া সিপিইউ টাইম। অর্থাৎ শুধু প্রোগ্রামটা এক্সিকিউট হতে যে সিপিইউ টাইম লাগে, সে সময়টা। I/O ওয়েটিং টাইমের মধ্যে পড়ে না।
* সিস্টেম টাইম (System time): সিস্টেম টাইম হলো, কার্নেলের নির্দিষ্ট কিছু ফাংশনের মধ্যে খরচ হওয়া সিপিইউ টাইম। অর্থাৎ কার্নেলের ভেতরে সিস্টেম কলে খরচ হওয়া সিপিইউ টাইম।
* ইলাপসড টাইম (Elapsed time): এটা হলো আসল, একেবারে অরিজিনালি খরচ হওয়া ওয়াল ক্লক টাইম। প্রোগ্রাম শুরু হওয়া থেকে একেবারে শেষ পর্যন্ত- অন্যান্য প্রসেসের টাইম স্লাইস, I/O ওয়েটিং টাইম- সবকিছু মিলেই ইলাপসড টাইম।

এসব উপাত্ত থেকে প্রোগ্রামের পারফরম্যান্স সম্পর্কে তথ্য পাওয়া যায়। যদি System time ও User time-এর সমষ্টি Elapsed time-এর চেয়ে বেশি ছোট হয় তাহলে ধরে নেওয়া I/Oতে প্রোগ্রামের পারফরম্যান্স ফল করছে।

### timeit

পাইথন স্ট্যান্ডার্ড লাইব্রেরির একটা মডিউল হলো টাইমইট। কমান্ডলাইন ইন্টারফেস ও পাইথন ইন্টারফেস দুটিই সাপোর্ট করে। পাইথনের ছোট ছোট কোড স্নিপেট কতটুকু সময় খাচ্ছে তা হিসাব করা যায় এই মডিউল দিয়ে। আর প্ল্যাটফর্ম স্পেসিফিক টাইম ফাংশন ব্যবহার করে বলে সবচেয়ে নির্ভুল ফলাফল পাওয়া যায়। উদাহরণ দেখা যাক:

python3 -m timeit '"-".join(str(n) for n in range(100))'

**আউটপুট**

10000 loops, best of 3: 41.5 usec per loop

এখানে python3-এর পরে -m আর্গুমেন্ট দিয়ে আমরা পাইথনকে বলছি টাইমইট মডিউলের খোঁজ নিতে ও এটাকে মেইন প্রোগ্রাম হিসেবে ব্যবহার করতে। -s আর্গুমেন্ট টাইমইট মডিউলে বলছে সেটআপকে একবার রান করতে। তারপর এটা আমাদের কোডকে রান করে। আউটপুট থেকে আমরা দেখতে পাচ্ছি যে আমাদের কোডকে তিনবার রান করা হয়েছে এবং বেস্ট অ্যাভারেজকে আউটপুট হিসেবে দেখানো হয়েছে। ওপরের এই একই কাজ আমরা পাইথন ইন্টারেক্টিভ শেলেও করতে পারি।

>>> import timeit

>>> timeit.timeit('"-".join(str(n) for n in range(100))', number=10000)

0.41126898900074593

আউটপুটে একটু হেরফের হয়েছে। আসলে কেবল কমান্ডলাইন ইন্টারফেসেই টাইমইট নিজ থেকে রিপিটেশনের সংখ্যা ঠিক করে নেয়।

### cProfile

পাইথন স্ট্যান্ডার্ড লাইব্রেরির আরেকটি বেঞ্চমার্কিং মডিউল হলো সিপ্রোফাইল। এটা আসলে একটা সি এক্সটেনশন (সাথে আরও কিছু)। আমাদের test.py ফাইলের প্রোগ্রামকে বেঞ্চমার্কিং করতে আমরা টার্মিনালে নিচের কমান্ডটা দেব।

python3 -m cProfile test.py

**আউটপুট**

100000 loops.

5 function calls in 0.010 seconds

Ordered by: standard name

ncalls tottime percall cumtime percall filename:lineno(function)

1 0.000 0.000 0.010 0.010 test.py:1(<module>)

1 0.010 0.010 0.010 0.010 test.py:1(ghoraghuri)

1 0.000 0.000 0.010 0.010 {built-in method builtins.exec}

1 0.000 0.000 0.000 0.000 {built-in method builtins.print}

1 0.000 0.000 0.000 0.000 {method 'disable' of '\_lsprof.Profiler' objects}

এর আউটপুটে বোঝার মতো কিছু বিষয় আছে। যেমন-

* ncalls: কল সংখ্যা।
* tottime: এই ফাংশনে মোট ব্যয়কৃত সময় (সাব-ফাংশন হিসেবের বাইরে)।
* percall: প্রত্যেক কলে ব্যয়কৃত সময়। tottime কে ncalls দিয়ে ভাগ দিলে পাওয়া যায়।
* cumtime: প্রোগ্রামের শুরু থেকে শেষ পর্যন্ত মোট ব্যয়কৃত সময় (সাব-ফাংশনসহ)।
* percall: cumtime কে ncalls দিয়ে ভাগ দিলে পাওয়া যায়।

### profilehooks

প্রোফাইলহুকসও একটি থার্ড-পার্টি মডিউল। এই মডিউলটি আসলে কয়েকটি প্রোফাইলিং ফাংশনের কালেকশন। আরও ক্লিয়ার করে বলতে গেলে পাইথন স্ট্যান্ডার্ড লাইব্রেরির টাইমইট এবং সিপ্রোফাইল মডিউলকে নিয়েই এখানের কাজগুলো। শুরুতেই ব্যবহার করার মডিউলটি আমরা ইনস্টল করব, সে জন্য নিচের মতো করে কমান্ড দেব-

sudo pip3 install profilehooks

ইনস্টল হয়ে গেল। এবার আমরা @profile ডেকোরেটরটাকে ব্যবহার করব। এ জন্য আমাদের test.py ফাইলের প্রোগ্রামটাকে প্রয়োজন অনুযায়ী রিরাইট করব।

from profilehooks import profile

@profile

def ghoraghuri():

count = 0

while count < 100000:

count += 1

return count

if \_\_name\_\_ == '\_\_main\_\_':

temp = ghoraghuri()

print(temp, 'loops.')

\*\* আউটপুট \*\*

100000 loops.

\*\*\* PROFILER RESULTS \*\*\*

ghoraghuri (test.py:3)

function called 1 times

2 function calls in 0.010 seconds

Ordered by: cumulative time, internal time, call count

ncalls tottime percall cumtime percall filename:lineno(function)

1 0.010 0.010 0.010 0.010 test.py:3(ghoraghuri)

1 0.000 0.000 0.000 0.000 {method 'disable' of '\_lsprof.Profiler' objects}

0 0.000 0.000 profile:0(profiler)

পরিচিত মনে হচ্ছে? এখানে সিপ্রোফাইল মডিউলটিকে ব্যবহার করা হয়েছে। এবার আমরা @timecall ডেকোরেটরটাকে ব্যবহার করব-

from profilehooks import timecall

@timecall

def ghoraghuri():

count = 0

while count < 100000:

count += 1

return count

if \_\_name\_\_ == '\_\_main\_\_':

temp = ghoraghuri()

print(temp, 'loops.')

**আউটপুট**

ghoraghuri (test.py:3):

0.009 seconds

100000 loops.

এই রেজাল্টটাও পরিচিত লাগছে? আসলে আমরা এখানে timeit মডিউলেরই ব্যবহার করেছি গোপনে। যাহোক, এবার আমরা @coverage ডেকোরেটরের ব্যবহার করব-

from profilehooks import coverage

@coverage

def ghoraghuri():

count = 0

while count < 100000:

count += 1

return count

if \_\_name\_\_ == '\_\_main\_\_':

temp = ghoraghuri()

print(temp, 'loops.')

**আউটপুট**

100000 loops.

\*\*\* COVERAGE RESULTS \*\*\*

ghoraghuri (test.py:3)

function called 1 times

@coverage

def ghoraghuri():

1: count = 0

100001: while count < 100000:

100000: count += 1

1: return count

এই হলো এই মডিউলের ব্যবহার। বিস্তারিতভাবে জানার জন্য <https://github.com/mgedmin/profilehooks> —এই লিঙ্কে ঘুরে আসতে পারেন।

### line\_profiler

এটি পাইথনের একটি থার্ডপার্টি মডিউল। এটি ব্যবহার করে খুব সহজেই একটি প্রোগ্রামের কোন লাইন কত সময় খাচ্ছে, তা দেখা যায়। প্রথমেই ইনস্টল করে নেওয়া যাক মডিউলটা।

sudo pip3 install line\_profiler

এবার আমরা আমাদের test.py ফাইলের বেঞ্চমার্কিং করব। তবে তার আগে test.py ফাইলের প্রোগ্রামটাকে নিচের মতো রিরাইট করতে হবে-

@profile

def ghoraghuri():

count = 0

while count < 100000:

count += 1

return count

if \_\_name\_\_ == '\_\_main\_\_':

temp = ghoraghuri()

print(temp, 'loops.')

আমরা যেসব ফাংশনের চুলচেরা বিশ্লেষণ করব, তাদের আগে @profile ডেকোরেটর ব্যবহার করতে হবে। এবার টার্মিনালে কমান্ড চালানো যাক-

kernprof -l -v test.py

**আউটপুট**

100000 loops.

Wrote profile results to test.py.lprof

Timer unit: 1e-06 s

Total time: 0.111737 s

File: test.py

Function: ghoraghuri at line 1

Line# Hits Time Per Hit %Time Line Contents

===============================================================

1 @profile

2 def ghoraghuri():

3 1 2 2.0 0.0 count = 0

4 100001 55342 0.6 49.5 while count < 100000:

5 100000 56392 0.6 50.5 count += 1

6 1 1 1.0 0.0 return count

ক্ষেত্রগুলো একটু ব্যাখ্যা করা যাক।

* Line#: কোডের লাইন নম্বর, যেটাকে চুলচেরা বিশ্লেষণ করা হয়েছে।
* Hits: যে কয়বার এই লাইনটি এক্সিকিউট হয়েছে।
* Time: লাইনটি এক্সিকিউট হতে মোট যতক্ষণ সময় নিয়েছে।
* Per Hit: লাইনটি এক্সিকিউট হতে গড়ে প্রতিবার যত সময় নিয়েছে।
* % Time: টোটাল সময়ের পরিপ্রেক্ষিতে শতকরা কতক্ষণ লাইনটা এক্সিকিউট হয়েছে।

### memory\_profiler

মেমোরি প্রোফাইলারও একটি থার্ড-পার্টি মডিউল যা মেমোরি কনজাম্পশন করার জন্য ব্যবহৃত হয়। এই মডিউল ব্যবহার করে লাইন বাই লাইন পাইথন প্রোগ্রামের মেমোরি কনজাম্পশন অ্যানালাইজ করা যায়। সুতরাং প্রথমেই মডিউলটি ইনস্টল করে নেওয়া যাক-

sudo pip3 install memory\_profiler

sudo pip3 install psutil

ইনস্টল হয়ে গেল। লাইন প্রোফাইলারের ক্ষেত্রে আমরা যেভাবে আমাদের প্রোগ্রামটা রিরাইট করেছিলাম, এখানে ঠিক একইভাবে @profile ডেকোরেটর বসাতে হবে ফাংশনের আগে। যেহেতু আমরা ইতিমধ্যে test.py ফাইলে আমাদের প্রোগ্রাম রিরাইট করে রেখেছি তাই আমরা সরাসরি অ্যানালিসিসে চলে যাব।

python3 -m memory\_profiler test.py

**আউটপুট**

100000 loops.

Filename: test.py

Line # Mem usage Increment Line Contents

===============================================================

1 31.648 MiB 0.000 MiB @profile

2 def ghoraghuri():

3 31.648 MiB 0.000 MiB count = 0

4 31.648 MiB 0.000 MiB while count < 100000:

5 31.648 MiB 0.000 MiB count += 1

6 31.648 MiB 0.000 MiB return count

এই মডিউলটির আরও অনেক ফাংশনালিটি আছে। বিস্তারিত জানার জন্য <https://github.com/fabianp/memory_profiler> এই লিঙ্কে ঘুরে আসতে পারেন।

### pycallgraph

কমান্ডলাইনে আউটপুট তো অনেক পেলাম। এবার একটু ভিজ্যুলাইজেশন যাওয়া যাক। এ জন্য আমরা পাইকলগ্রাফ মডিউলটি ব্যবহার করব।

sudo pip3 install pycallgraph

sudo apt-get install graphviz -y

ইনস্টল হয়ে গেলে এবার আমরা নিচের কমান্ডটা দেব-

pycallgraph graphviz -- ./test.py

কিন্তু এখানে শুধু প্রোগ্রামের আউটপুটটি দেখতে পাব, কোনো বেঞ্চমার্ক ইনফরমেশন দেখতে পাব না। দুঃখ পাওয়ার কিছু নেই। test.py-এর ডিরেক্টরিতে গিয়ে দেখব pycallgraph.png নামে কোনো ফাইল আছে কি না। এই ফাইলটা ওপেন করলেই আমরা সাত রাজার ধন পেয়ে যাব।

আরও জানার জন্য পাইকলগ্রাফের অফিশিয়াল ডকুমেন্টেশন- <http://pycallgraph.slowchop.com/en/master/>

## মেশিন লার্নিং, আর্টিফিশিয়াল ইন্টিলিজেন্স, ডীপ লার্নিং, টেনসরফ্লো ইত্যাদির একটার সাথে আরেকটার সম্পর্ক

মেশিন লার্নিং, আর্টিফিশিয়াল ইন্টিলিজেন্স, ডীপ লার্নিং, টেনসরফ্লো ইত্যাদির একটার সাথে একটার সম্পর্ক এবং মেশিন লার্নিং কিভাবে শেখা যাবে, শিখে কোথায় প্রয়োগ করা যাবে, তা নিয়ে বিস্তারিত 🙂

### মেশিন লার্নিং:

টারমিনেটর মুভির কথা মনে আছে? স্কাইনেট এর মত কোন কিছুর উৎপত্তি যদি হয়, তা হবে এই মেশিন লার্নিং থেকে। মানুষ থেকে বেশি বুদ্ধিমান প্রোগ্রাম হলে যা হবে আরকি।

কোন কিছু বুদ্ধিমান হতে হলে তাকে শিখতে হয়। কোন প্রোগ্রাম নিজে নিজে শিখতে পারাটা হচ্ছে মেশিন লার্নিং। এটা রোবট হতে পারে বা যে কোন সফটওয়ার হতে পারে। সাধারণ কম্পিউটার প্রোগ্রামে কোনটার পর কি কাজ করবে সে লজিক গুলো বলে দেই, প্রোগ্রাম সে অনুযায়ী কাজ করে। কিন্তু মেশিন লার্নিং প্রোগ্রাম গুলো একটু ভিন্ন। এগুলো নিজে থেকেই বুঝে নেয় কখন কি কাজ করতে হবে। এখানে প্রোগ্রাম গুলোকে আগে কিছু ডেটা দিয়ে ট্রেইন করা হয়। প্রথমে এরা সঠিক আউটপুট দেয় না। শিখতে শিখতে এক সময় সঠিক আউটপুট দেয়।

এরকম প্রোগ্রাম গুলো আমরা নিয়মিত ব্যবহার করেছি। যেমন ফেসবুকের কথাই চিন্তা করি, আপনার নিউজ ফিডে আপনি সবার পোস্ট দেখতে পান না। ফেসবুক অ্যালগরিদম আপনার পছন্দ অনুযায়ী আপনাকে নিউজ দেখায়। আপনি কি পছন্দ করেন না করেন, তা এনালাইসিস করে ফেসবুক আস্তে আস্তে বুঝে নেয় আপনাকে কি দেখাতে হবে। এটা একটা উদাহরণ। আমাদের স্মার্ট ফোনে অনেক গুলো মেশিন লার্নিং প্রোগ্রাম রয়েছে। সিরি, কর্টনা, ওকে গুগল, ভয়েস দিয়ে সার্চ, ফটো অ্যাপ ইত্যাদি সব গুলতেই মেশিন লার্নিং এর প্রয়োগ রয়েছে।

### Types of Machine Learning:

মেশিন লার্নিং হচ্ছে আসলে কিছু অ্যালগরিদম। মেশিন লার্নিং অ্যালগরিদম গুলোকে সাধারণত চার ভাগে ভাগ করা যায়।

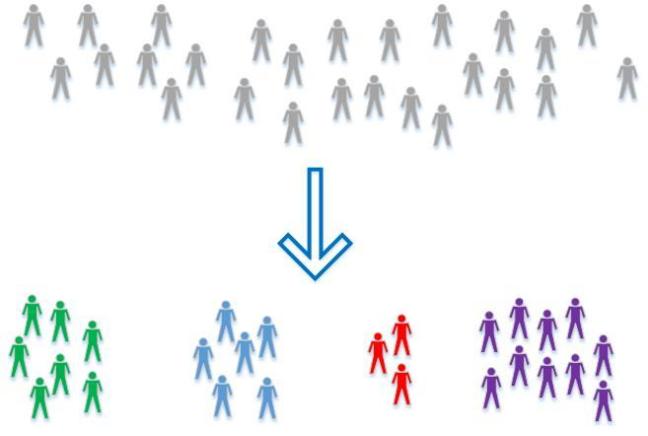
* Supervised learning
* Unsupervised learning
* Semi-Supervised learning
* Reinforcement learning

### সুপারভাইসড লার্নিং

সুপারভাইসড লার্নিং এ গুলো উদাহরণ বা ট্রেনিং ডেটা থেকে প্রোগ্রাম বুঝে নেয় তার কি করতে হবে। প্রথমে কিছু ট্রেনিং ডেটা দিয়ে প্রোগ্রামকে ট্রেইন করা হয়। এই ট্রেনিং ডেটাতে ইনপুট দিলে কি আউটপুট হবে, তা থাকে। ট্রেইন করা হয়ে গেলে প্রোগ্রাম নিজ থেকেই বুঝে নেয় তার কি করা উচিত। ছোট্ট একটা উদাহরণ দেওয়া যেতে পারে।

1 ইনপুট দিলে আউটপুট দিবে 1। দুই ইনপুট দিলে আউটপুট দিবে 4, তিন ইনপুট দিলে আউটপুট দিবে 9। এভাবে ট্রেনিং ডেটা এনালাইসিস করে প্রোগ্রাম বুঝে কোন নাম্বার ইনপুট দিলে আউটপুট দিতে হবে ঐ নাম্বারটির বর্গ। তারপর আমরা যখনই কোন নাম্বার দিব, প্রোগ্রামটি আমাদের ঐ নাম্বারের বর্গ আউটপুট দিবে। কোন কম্পিউটার প্রোগ্রামকে এভাবে শিখিয়ে নেওয়াটাই হচ্ছে সুপারভাইসড লার্নিং।

### আনসুপারভাইসড লার্নিং

[](https://jakir.me/%e0%a6%ae%e0%a7%87%e0%a6%b6%e0%a6%bf%e0%a6%a8-%e0%a6%b2%e0%a6%be%e0%a6%b0%e0%a7%8d%e0%a6%a8%e0%a6%bf%e0%a6%82-%e0%a6%9f%e0%a7%87%e0%a6%a8%e0%a6%b8%e0%a6%b0%e0%a6%ab%e0%a7%8d%e0%a6%b2%e0%a7%8b/customer_segmentation_vert)

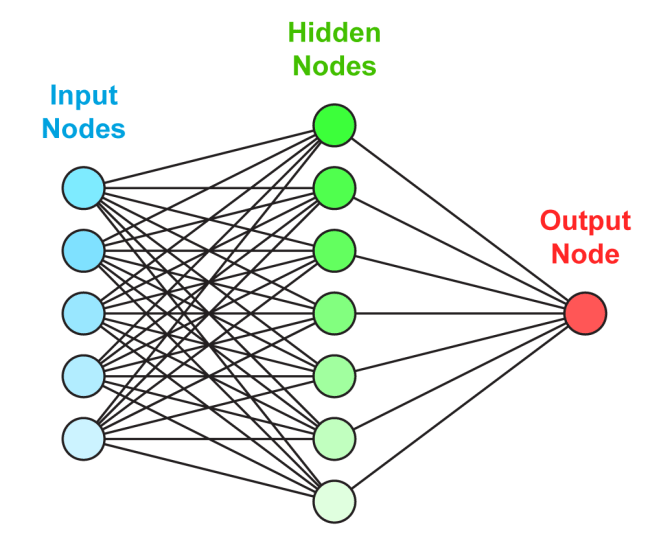
আনসুপারভাইসড লার্নিং এ ট্রেনিং ডেটার আউটপুট কি হবে, তা বলে দেওয়া হয় না। ইনপুট দিলে প্রোগ্রাম নিজ থেকেই বুঝে নেয় কি করতে হবে। যেমন আমরা যদি অনেক গুলো মানুষের তথ্য ইনপুট দেই, প্রোগ্রামটি তখন তথ্য গুলোর মধ্যে সিমিলারিটি খুঁজে সিমিলার মানুষ গুলোকে এক সাথে রাখবে। বিভিন্ন ক্লাসে বা ক্লাস্টারে ভাগ করবে। এটা হচ্ছে আনসুপারভাইসড লার্নিং এর একটা উদাহরণ।

### রিইনফোর্সমেন্ট লার্নিং

আমরা ছোটবেলায় কিভাবে শিখি? কোন কিছু করার পর ভালো লাগলে তা বেশি করে করি। আবার কোন কিছু করে ব্যথা পেলে তা আর করি না। বাস্তবে মানুষ বা কোন প্রাণী যেভাবে শিখে, রিইনফোর্সমেন্ট লার্নিং এ কোন প্রোগ্রামে ঠিক ঐ ভাবে ট্রেইন করা হয়।

### নিউরাল নেটওয়ার্ক

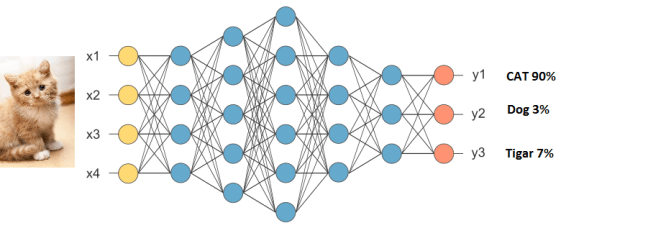
মানুষের ব্রেইন যেভাবে কাজ করে, তাকে অনুকরণ করার প্রচেষ্ঠা থেকেই আর্টিফিশিয়াল নিউরাল নেটোওয়ার্ক শাখাটির উৎপত্তি। মানুষের নিউরনের মত করে এখানে থাকে নড। নড গুলোতে কোন ডেটা আসলে সেখানে অপারেশন চালিয়ে তা পরবর্তী নডে পাস করে দেয়। নড গুলো একটা আরেকটার সাথে কানেক্টেড থাকে, এগুলোকে বলে লিঙ্ক। প্রতিটা লিঙ্ক এ একটা ওয়েট থাকে। আর প্রতিবার শেখার পড় এই ওয়েট পরিবর্তন করে।

[](https://jakir.me/%e0%a6%ae%e0%a7%87%e0%a6%b6%e0%a6%bf%e0%a6%a8-%e0%a6%b2%e0%a6%be%e0%a6%b0%e0%a7%8d%e0%a6%a8%e0%a6%bf%e0%a6%82-%e0%a6%9f%e0%a7%87%e0%a6%a8%e0%a6%b8%e0%a6%b0%e0%a6%ab%e0%a7%8d%e0%a6%b2%e0%a7%8b/nn)

নিউরাল নেটওয়ার্ক ব্যবহার করে যে কোন ধরণের মেশিং লার্নিং অ্যালগরিদম যেমন সুপারভাইসড লার্নিং, আনসুপারভাইসড লার্নিং, রিইনফোর্সমেন্ট লার্নিং, ইত্যাদি প্রয়গো করা যায়।

### ডীপ লার্নিং

মাল্টি লেয়ার নিউরয়াল নেটোওয়ার্ক হচ্ছে ডীপ লার্নিং বা ডীপ নিউলার নেটওয়ার্ক লার্নিং। সাধারণ নিউরাল নেটওয়ার্কে থাকে ইনপুট লেয়ার, একটা হিডেন লেয়ার আর আউটপুট লেয়ার। ডীপ লার্নিং এ থাকে একের অধিক হিডেন লেয়ার। আর এ জন্যই এর নাম ডীপ লার্নিং।

[](https://jakir.me/%e0%a6%ae%e0%a7%87%e0%a6%b6%e0%a6%bf%e0%a6%a8-%e0%a6%b2%e0%a6%be%e0%a6%b0%e0%a7%8d%e0%a6%a8%e0%a6%bf%e0%a6%82-%e0%a6%9f%e0%a7%87%e0%a6%a8%e0%a6%b8%e0%a6%b0%e0%a6%ab%e0%a7%8d%e0%a6%b2%e0%a7%8b/deep_neural_network)

ডীপ লার্নিং ও নিউরাল নেটওয়ার্ক নিয়ে অনেক আগে থেকেই কাজ হচ্ছিল। কিন্তু আগে এত জনপ্রিয় হতে পারে নি বা মানুষ তত বেশি ব্যবহার করে নি। এখন ডীপ লার্নি একটা হট টপিক। আগে ডীপ লার্নিং নিয়ে কাজ করা সহজ না থাকার কারণ ছিল কম্পিউটেশনাল পাওয়ার। আমাদের সিপিইউ কোন ডেটা লিনিয়ার প্রসেস করে। ডীপ লার্নিং এ অনেক অনেক কম্পিউটেশন করতে হয়। আর ডীপ লার্নিং নিয়ে কাজ করতে তাই অনেক বেশি সময় লাগত। জিপিইউ প্যারালালি ডেটা প্রসেস করতে পারে। জিপিইউ এর ক্ষমতা ব্যবহার করে ডীপ লার্নিং ব্যবহার করে মেশিন লার্নিং এ দ্রুত ফলাফল পাওয়ার কারণে ডীপ লার্নিং এর ব্যবহার দিন দিন বাড়তে লাগল। ডীপ লার্নিং এর ব্যবহার এতই বেশি হচ্ছে যে এখন মেশিন বলতেই ডীপ লার্নিং বুঝে অনেকে।

ডীপ লার্নিং হচ্ছে এখানেঃ Computer Science > Artificial Intelligence > Machine Learning > Neural Network > Deep Learning। এত গভীরে হওয়ার কারণেই সম্ভবত এর নাম ডীপ লার্নিং 😛

### মেশিন লার্নিং এর জন্য প্রোগ্রামিং ল্যাঙ্গুয়েজ

মেশিন লার্নিং হচ্ছে কত গুলো অ্যালগরিদমের সমষ্টি। আমরা এই অ্যালগরিদম গুলো যে কোন ল্যাঙ্গুয়েজে ইমপ্লিমেন্ট করে মেশিন লার্নিং নিয়ে কাজ করতে পারি। কিছু কিছু প্রোগ্রামিং ল্যাঙ্গুয়েজে বিল্ট ইন কিছু অ্যালগরিদম ইমপ্লিমেন্টেড থাকার কারণে ঐসব ল্যাঙ্গুয়েজ বেশি জনপ্রিয়। যেমন R প্রোগ্রামিং ল্যাঙ্গুয়েজ। এটি মূলত স্ট্যাটিসটিক্স রিলেটেড কাজ করার জন্য ডেভেলপ করা হয়েছে। ডেভেলপ ও করেছেন স্ট্যাটিস্টিশিয়ানরা। মেশিন লার্নিং এর অনেক অংশ জুড়ে ডেটা নিয়ে কাজ করতে হয়। আর এ জন্যই R প্রোগ্রামিং ল্যাঙ্গুয়েজ মেশিন লার্নিং এ অনেক জনপ্রিয়।

এ ছাড়া পাইথনের অনেক গুলো লাইব্রেরি রয়েছে ডেটা নিয়ে কাজ করার জন্য। এগুলোর মধ্যে scikit-learn অনেক বেশি ব্যবহার করা হয়। পাইথন প্রোগ্রামিং অনেক সহজ এবং দারুণ প্রোগ্রামিং হওয়ায় এটিও মেশিন লার্নিং এর জন্য প্রচুর ব্যবহৃত হয়।

### মেশিন লার্নিং লাইব্রেরী ও ফ্রেমওয়ার্ক

মেশিন লার্নিং এর মধ্যে অনেক গুলো শাখা প্রশাখা রয়েছে। এক একটা কাজের জন্য এক এক ধরণের অ্যালগরিদম ব্যবহার করা হয়। যেমন কম্পিউটার ভিশন এর জন্য এক ধরণের অ্যালগরিদম, ন্যচারাল ল্যাঙ্গুয়েজ প্রসেসিং এর জন্য এক ধরণের অ্যালগরিদম, ভয়েজ রিকগনিশনের জন্য এক ধরণের অ্যালগরিদমের প্রয়োজন হয়। লারিব্রেরী বা ফ্রেমওয়ার্ক গুলোতে একই ক্যাটেগরির দরকারি অ্যালগরিদম গুলো ইমপ্লিমেন্ট করা থাকে। যেমন ন্যাচারাল ল্যাঙ্গুয়েজ প্রসেসিং করতে যে ধরণের অ্যালগরিদম গুলো দরকার হতে পারে, সব গুলো ইমপ্লিমেন্ট করে একটা লাইব্রেরী তৈরি করা হয়। এরপর যে কেউ ঐ লাইব্রেরী নিজের মত করে ব্যবহার করতে পারে।

এরকম সব গুলো লাইব্রেরীর একটা লিস্ট পাওয়া যাবে গিটহাবের [**Awesome Machine Learning**](https://github.com/josephmisiti/awesome-machine-learning) নামক রিপোজিটরিতে। সেখানে সব গুলো ল্যাঙ্গুয়েজ অনুযায়ী লাইব্রেরী গুলোর লিস্ট করা রয়েছে। কোন লাইব্রেরী দিয়ে কি ধরনের কাজ করা হবে, সে সব তথ্যও পাওয়া যাবে। সেগুলো দেখে নিজ প্রয়োজন মত যে কোন লাইব্রেরী নিয়েই কাজ করতে পারেন।

### টেনসরফ্লো

[**টেনসরফ্লো**](https://www.tensorflow.org/) হচ্ছে ডীপ লার্নিং এর জন্য একটা লাইব্রেরী। গুগল ব্রেইন টিম এটি ডেভেলপ করেছে। এটা শুরু করার উদ্দেশ্য ছিল মূলত রিসার্চ এর কাজে ব্যবহার করার জন্য। পরে সবার জন্য উন্মুক্ত করে দেওয়া হয়।

গুগল নিজের প্রোডাক্ট তৈরি করতে যে প্রযুক্তি ব্যবহার করে, তা আমরা যে কেউই ব্যবহার করতে পারি। ছোট একটা কোম্পানি বা কোন পারসন আর গুগলের মধ্যে গ্যাপ টা কত কম। গুগলের স্পিচ রিকগনিশন সিস্টেম, গুগল ফটো, জিমেইল, সার্চ এসব কিছুতে টেনসরফ্লো বা ডীপ লার্নিং ব্যবহার করা হয়। গুগল যা করতে পারে, আমরাও তা করতে পারি। তৈরি করতে পারি দারুণ কিছু।

টেনসরফ্লো নিয়ে কাজ করার জন্য রয়েছে Python API। টেনসরফ্লো এর বেশির ভাগ কোড লেখা হয়েছে C++ এ। C++ API ও রয়েছে। তবে পাইথনের ডকুমেন্টশন বেশি। পাইথনে কোড লেখাও সহজ। নতুন যারা টেনসরফ্লো নিয়ে আগ্রহী, তারা পাইথন দিয়ে শুরু ক্রতে পারেন। টেনসরফ্লো এর রিসোর্স এবং ওপেন সোর্স প্রজেক্ট গুলোর একটা লিস্ট পাওয়া যাবে [**Awesome TensorFlow**](https://github.com/jtoy/awesome-tensorflow)রিপোজিটোরিতে।

### মেশিন লার্নিং নিয়ে কাজ করার জন্য সহজ কিছু প্লাটফর্ম

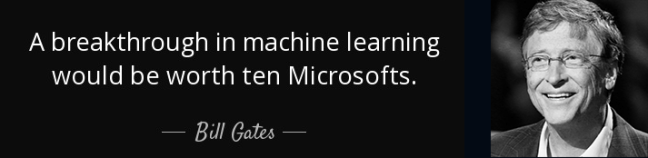
যারা কোডিং ব্যাকগ্রাউণ্ড এর না বা সহজে দারুণ কিছু তৈরি করতে চান, তারা দেখতে পারেন [**wit.ai**](https://wit.ai/), [**IBM Watson API**](https://www.ibm.com/watson/) বা [**Google Prediction**](https://cloud.google.com/prediction/) প্লাটফরম গুলো। এগুলোতে সব কিছুই করা আছে। আপনি শুধু API কল আপনার প্রোগ্রামে ব্যবহার করতে পারবেন।

### কোথায় থেকে শেখা যাবে

ডীপ লার্নিং এর নামের সাথে একজনের নাম চলে আসে, তার নাম Andrew NG। [**কোর্সেরাতে মেশিন লার্নিং**](https://www.coursera.org/learn/machine-learning) এর উপর উনার করা কোর্স রয়েছে। এছাড়া উডাসিটিতে [**ডীপ লার্নিং**](https://www.udacity.com/course/deep-learning--ud730) এবং টেনসরফ্লো নিয়ে গুগলের নিজস্ব কোর্স রয়েছে। এগুলো শেখার জন্য দারুণ রিসোর্স। অনলাইনে আরো অনেক রিসোর্স রয়েছে।

গ্রাউন্ড লেভেল থেকে মেশিন লার্নিং নিয়ে কাজ করার জন্য গুগলেরই তৈরি [**Machine Learning Recipes**](https://www.youtube.com/playlist?list=PLOU2XLYxmsIIuiBfYad6rFYQU_jL2ryal) সিরিজটা খুবি সহজ এবং দারুণ। আমার নিজের কাছে অনেক বেশি ভালো লেগেছে। এছাড়া আরো রিসোর্স খুঁজে নেওয়ার জন্য গুগল তো আছেই।

### কোথায় এপ্লাই করা যাবে

[](https://jakir.me/%e0%a6%ae%e0%a7%87%e0%a6%b6%e0%a6%bf%e0%a6%a8-%e0%a6%b2%e0%a6%be%e0%a6%b0%e0%a7%8d%e0%a6%a8%e0%a6%bf%e0%a6%82-%e0%a6%9f%e0%a7%87%e0%a6%a8%e0%a6%b8%e0%a6%b0%e0%a6%ab%e0%a7%8d%e0%a6%b2%e0%a7%8b/bill-gates-about-machine-learning)

সব জায়গায় মেশিং লার্নিং এপ্লাই করা যাবে। আমরা যত গুলো অ্যাপ ব্যবহার করি, যত গুলো সফটওয়ার ব্যবহার করি, কয়েক বছরের মধ্যে সব গুলো সফটওয়ার, অ্যাপ, ওয়েব সাইট মেশিন লার্নিং প্রোগ্রাম দিয়ে রিপ্লেস হয়ে যাবে। ভবিষ্যৎ এ ভালো কিছু করতে চাইলে মেশিন লার্নিং নিয়ে পড়ালেখা একটা ভালো অপশন হতে পারে।

IoT এবং মেশিং লার্নিং এর কম্বিনেশনে হবে দারুণ কিছু। [ইন্ডিয়াতে প্রায় 470+ লিস্টেড IoT স্টার্টআপ রয়েছে](https://www.linkedin.com/pulse/internet-things-iot-startups-india-jan-2017-pal-choudhury)। সারা বিশ্বে রয়েছে 3000+।

### মেশিন লার্নিং এবং ভবিষ্যৎ

আমরা প্রোগ্রামাররা এখন যেমন পাইথন, জাভা, জাভাস্ক্রিপ্ট ইত্যাদি শিখি, কয়েক বছরের মধ্যে মেশিন লার্নিং তেমন ভাবে শিখব। মানে সাধারণ প্রোগ্রামারদেরও Machine Learning সম্পর্কে জানতে হবে। আসলে কয়েক বছর নয়, এখনই মেশিন লার্নিং এর চাহিদা সবচেয়ে বেশি।  
সাধারণ প্রোগ্রামে আমরা বলে দেই কি কি কাজ করবে একটা প্রোগ্রাম। Machine Learning প্রোগ্রাম গুলোকে আমরা ডেটা দেই , ঐ ডেটা থেকে প্রোগ্রাম নিজের মত করে প্রোগ্রাম তৈরি করে নেয়। এটাই পার্থক্য। দারুণ সব সমস্যা সমাধাণ আমরা করে ফেলতে পারি মেশিন লার্নিং এর মাধ্যমে। [মেশিন লার্নিং সম্পর্কে বিস্তারিত একটা লেখা।](https://jakir.me/%E0%A6%AE%E0%A7%87%E0%A6%B6%E0%A6%BF%E0%A6%A8-%E0%A6%B2%E0%A6%BE%E0%A6%B0%E0%A7%8D%E0%A6%A8%E0%A6%BF%E0%A6%82)  
মনে হতে পারে মেশিন লার্নিং কঠিক কিছু, আসলে কঠিন না। যার প্রোগ্রামিং সম্পর্কে ধারণা রয়েছে, রয়েছে মাধ্যমিকে পড়ে আসা গণিত সম্পর্কে ধারণা সেই শিখতে পারে। সেখার জন্য অনলাইনে প্রচুর টিউটোরিয়াল রয়েছে।  
Machine Learning সহজ করার জন্য অনেক গুলো লাইব্রেরী রয়েছে, যেমন পাইথনের জন্য scikit-learn, রয়েছে গুগলের TensorFlow, Apache Spark MLlib, মাইক্রোসফটের Azure ML Studio ইত্যাদি। Machine Learning এর থিওরি পার্ট হয়তো একটু কঠিন মনে হতে পারে, কিন্তু ব্যাসিক আইডিয়া নেওয়ার পর প্র্যাক্টিক্যাল কাজ করা অনেক সহজ করে তুলছে এসব লাইব্রেরী গুলো। এগুলো কমপ্লেক্স সব অ্যালগরিদম ইমপ্লিমেন্ট করে রেখেছে। আমদের শুধু ব্যবহার করতে হবে। শুধু জানতে হবে কোন অ্যালগরিদম বা কোন লাইব্রেরী নিজ প্রোগ্রামের জন্য বেস্ট।  
মেশিন লার্নিং এ ট্রেনিং ডেটা দিয়ে প্রোগ্রামকে ট্রেইন করতে হয়। অনলাইনে অনেক ডেটা রয়েছে। আবার অনেক ক্লাসিফায়ারও রয়েছে, যেগুলোকে ট্রেইনিং ডেটা দিয়ে ট্রেইন করা। যেমন ইমেজ নিয়ে কাজ করার জন্য রয়েছে Inception. এটা গুগলের বেস্ট ইমেজ ক্লাসিফায়ার। এটিকে ১.২ মিলিয়ন ইমেজ ডেটার উপর ট্রেইন করা হয়েছে। এত গুলো ইমেজ দিয়ে ইনসেপশনকে ট্রেইন করাতে দুই সপ্তাহ লেগেছে। আমরা সহজেই যে কোন ইমেজকে ক্লাসিফাই করতে ইনসেপশন ব্যবহার করতে পারি। এই টার্ম গুলো না বুঝলে এই লেখাটি দেখতে পারেনঃ [**মেশিন লার্নিং এর কিছু টার্ম**](https://jakir.me/%E0%A6%AE%E0%A7%87%E0%A6%B6%E0%A6%BF%E0%A6%A8-%E0%A6%B2%E0%A6%BE%E0%A6%B0%E0%A7%8D%E0%A6%A8%E0%A6%BF%E0%A6%82-%E0%A6%8F%E0%A6%B0-%E0%A6%9F%E0%A6%BE%E0%A6%B0%E0%A7%8D%E0%A6%AE)  
গুগলেরেই অনেক গুলো টিউটোরিয়াল রয়েছে মেশিন লার্নিং এর উপর। এমনকি এ জন্য গুগল যে টুল গুলো ব্যবহার করে, তার বেশিরভাগই ওপেনসোর্স। গুগলের [**Machine Learning Recipes**](https://www.youtube.com/playlist?list=PLOU2XLYxmsIIuiBfYad6rFYQU_jL2ryal) নামে কয়েকটি টিউটোরিয়াল রয়েছে । খুবি দারুণ। এছাড়া ইউডাসিটিতে [**ডীপ লার্নিং**](https://www.udacity.com/course/deep-learning--ud730?utm_campaign=chrome_series_machinelearning_063016&utm_source=gdev&utm_medium=yt-desc) এর উপর গুগলের ফ্রি কোর্সটাও দারুণ।  
TopCoder, UVA এর মত ডেটা সাইন্স এবং মেশিন লার্নিং এর দারুণ একটা প্লাটফর্ম হচ্ছে [**Kaggle**](https://kaggle.com/) । এখানে ডেটা সাইন্স  এর উপর কন্টেস্ট হয়। রয়েছে অনেক গুলো পাবলিক ডেটা। যেগুলো ব্যবহার করে Machine Learning এবং ডেটা সাইন্স শেখা যায়। আবার এই ডেটা কে কিভাবে ব্যবহার করেছে, তার সোর্স কোড সাবমিটও করা যায়। অন্যরা একটা সমস্যা কিভাবে সমাধান করেছে, তা দেখা যায়। দেখে দেখে শেখা যায়। শেখার জন্য উৎকৃষ্ট একটা প্লাটফর্ম। ইন্টারনেটের আনাচে কানাচে আরো অনেক রিসোর্স রয়েছে শেখার জন্য। গুগল, ফেসবুকের মত কম্পানি সহ ছোট বড় সব কম্পানিতে মেশিন লার্নিং এক্সপার্টদের জন্য অনেক গুলো জব পড়ে রয়েছে। লিঙ্কডইনে একটু ঢু মারলেই দেখতে পাবেন। তাই শিখে কি হবে, তা চিন্তা না করে যদি ভালো লাগে, শিখতে পারেন

### মেশিন লার্নিং এর কিছু টার্ম

**মেশিন লার্নিং:** কম্পিউটার সাইন্সের একটা শাখা হচ্ছে আর্টিফিশিয়াল ইন্টেলিজেন্স। আর্টিফিশিয়াল ইন্টেলিজেন্স এর একটি শাখা হচ্ছে এই মেশিন লার্নিং। মেশিন লার্নিং এর কাজ হচ্ছে ডেটার উপর ভিত্তি করে ডিসিশন দেওয়া এবং প্রিডিক্ট করা।

**Supervised Learning:** কিছু প্রি ডিফাইন ডেটাসেট এর উপর প্রোগ্রামকে ট্রেইন করা হয়। ঐ ট্রেইন ডেটা এর উপর ভিত্তি করে প্রোগ্রাম ডিসিশন দেয়। এটা হচ্ছে সুপারভাইসড লার্নিং। যেমন মেইলটি কি স্প্যাম না কি স্প্যাম না, এই ডিসিশনটা আগের কিছু ডেটার উপর নির্ভর করে দেয়া হয়। এটা হচ্ছে সুপারভাইসড লার্নিং এর উদাহরণ।

**Unsupervised Learning:** আনসুপারভাইসড লার্নিং এ প্রোগ্রামকে কিছু ডেটা দেওয়া হয়। প্রোগ্রাম ঐ ডেটার উপর নির্ভর করে ডিসিশন দেয়। যেমন এক ঝুড়ি ফল রয়েছে। প্রোগ্রাম ভিন্ন ভিন্ন ফল কে ভিন্ন ভিন্ন ক্যাটেগরিতে ভাগ করবে, এটা হচ্ছে আনসুপারভাইসড লার্নিং এর উদাহরণ।

**Classification:** সুপারভাইসড লার্নিং এর একটি শাখা হচ্ছে হচ্ছে ক্লাসিফিকেশন। নাম থেকেই বুঝা যায় এর কাজ কি। এর কাজ হচ্ছে ইনপুট ডেটাকে বিভিন্ন ক্লাসে ভাগ করা। সাধারণত Yes or No এর ভিত্তিতে ডেটাকে ক্লাসিফিকেশন করা হয়। যেমন আপনার কাছে একটা ছবি আছে। ঐ ছবিটা ছেলের ছবি নাকি মেয়ের ছবি, এটার ম্যাপিং হচ্ছে ক্লাসিফিকেশন। বিচ্ছিন্ন ডেটার জন্য ক্লসিফিকেশন ব্যবহার করা হয়।

**Regression:** সুপারভাইসড লার্নিং এর আরেকটি শাখা হচ্ছে রিগ্রেশন। যেখানে Yes or No দিয়ে ডেটা প্রিডিকশন করা যায় না, সেখানে রিগ্রেশন ব্যবহার করা হয়। How much?, How many? টাইপ প্রশ্ন গুলোর উত্তর রিগ্রেশন দিয়ে বের করা হয়। কন্টিনিউয়াস ডেটার জন্য রিগ্রেশন ব্যবহার করা হয়।

**Decision Tree:** Tree এর মত গ্রাফ মডেল দিয়ে ডিসিশন নেওয়ার জন্য Decision Tree ব্যবহৃত হয়। অনেকটা ফ্লো চার্ট এর মত।

**Deep Learning:** মেশিন লার্নিং ইমপ্লিমেন্টেশনের একটা টেকনিক হচ্ছে ডীপ লার্নিং। যেখানে আর্টিফিশিয়াল নিউরাল নেটোওয়ার্ক ব্যবহার করে ডেটার উপর ডিসিশন নেওয়া হয়। আমরা অনেকেই প্রিজসমা অ্যাপটি ব্যবহার করেছি। এটি অনেক সুন্দর ভাবে ছবিকে আর্টে পরিণত করে দেয়। ঐ ছবিকে আর্টে পরিণত করার জন্য ডীপ লার্নিং ব্যবহার করা হচ্ছে। ডীপ লার্নিং মেশিন লার্নিং এর খুবই জনপ্রিয় ক্যাটেগরি।

মেশিন লার্নিং, ডেটা এনালাইসিস, আর্টিফিশিয়াল ইন্টেলিজেন্স এগুলো শেখার জন্য খুবি জনপ্রিয় বিষয়। এখন শেখা শুরু করলে ভবিষ্যৎ এ অনেক ভালো করার সুযোগ রয়েছে। শেখার জন্য অনলাইনে বিভিন্ন ইউনিভার্সিটির অনেক গুলো ফ্রি কোর্স রয়েছে। গুগলে একটু খুঁজলেই পাওয়া যাবে।

**আর্টিফিশিয়াল ইন্টিলিজেন্সঃ**

বর্তমান এবং ভবিষ্যৎ এর সবচেয়ে গুরুত্বপূর্ণ টপিক্স হচ্ছে এই আর্টিফিশিয়াল ইন্টিলিজেন্স।

মেশিনের বুদ্ধি শুদ্ধিকে আমরা বলি আর্টিফিশিয়াল ইন্টেলিজেন্স। আর বুদ্ধি হচ্ছে জ্ঞান আহরণ করা এবং তা প্রয়োগ করার ক্ষমতা। মেশিন জ্ঞান আহরণটা হচ্ছে মেশিন লার্নিং। দুইটা এক সাথে মানে শেখা এবং প্রয়োগ করাই হচ্ছে আর্টিফিশিয়াল ইন্টীলিজেন্স।

আমরা মুভি বা সাইন্স ফিকশনে দেখেছি রোবটরেরা অনেক শক্তিশালী হয়ে উঠে। মানুষকে ধ্বংস করে। কিছু রোবট মানুষের উপকারে কাজ করে। এই  রোবট গুলোর ব্রেইনটাই হচ্ছে আর্টিফিশিয়াল ইন্টীলিজেন্স। আর্টিফিশিয়াল ইন্টিলিজেন্স ছাড়া রোবট গুলো সাধারণ জড় পদার্থের মত। এমন না যে আর্টিফিশিয়াল ইন্টিলিজেন্স এর জন্য আমাদের রোবট লাগবে। আমাদের পকেটের যে স্মার্ট ফোনটা রয়েছে, এটাতেই অনেক গুলো আর্টিফিশিয়াল ইন্টিলিজেন্স যুক্ত অনেক গুলো অ্যাপ রয়েছে।

**আর্টিফিশিয়াল ইন্টিলিজেন্স এর ধাপঃ**

আর্টিফিশিয়াল ইন্টিলিজেন্স প্রোগ্রাম গুলোকে তিনটে ধাপে ভাগ করা যায়।

ANI or Artificial Narrow Intelligence or Weak AI. ANI প্রোগ্রাম গুলো একটা নির্দিষ্ট কাজে এক্সপার্ট হয়। যেমন যে প্রোগ্রামটি দাবা খেলতে পারে, সে দাবা খেলাই সবচেয়ে ভালো পারবে। তাকে অন্য কোন কাজ করতে দিলে সে কিছুই করতে পারবে না। যেমন যদি তাকে  লুডু খেলতে দেওয়া হয়, সে পারবে না। ছোট বাচ্চাদেরকে কঠিন কোন কিছু করতে বললে তারা যেভাবে তাকায়, প্রোগ্রামটি আপনার দিকে ঠিক সেভাবে তাকাবে।

AGI or Artificial General IntelligenceI:  কৃত্রিম বুদ্ধিমত্তার দ্বিতীয় ধাপ হচ্ছে AGI। এটাকে Strong AI, বা Human-Level AI ও বলা হয়। এ ধাপে কম্পিউটার মানুষের মত চিন্তা করতে পারবে, মানুষের মত প্ল্যান করা, সমস্যা সমাধান করা, হঠাৎ নতুন কোন পরিবেশে আসলে চারপাশ দেখে সে পরিবেশের সাথে মানিয়ে নেওয়ার মত ক্ষমতা অর্জা করবে।

ASI or Artificial Super Intelligence: কৃত্রিম বুদ্ধিমত্তার তৃতীয় ধাপ। কম্পিউটার যখন মানুষ থেকেও বুদ্ধিমান হবে তখন তাকে আমরা বলব আর্টিফিশিয়াল সুপার ইন্টেলিজেন্স। আর্টিফিশিয়াল সুপার ইন্টেলিজেন্স সিস্টেম তৈরি হলে কি হবে, মানুষের ভালো হবে না খারাপ হবে, তা নিয়ে গবেষকরা এখনই চিন্তিত।

Artificial Super Intelligence এ পৌঁছানোর পর আমরা থাকব কি থাকব না তা নিয়ে বিজ্ঞানীরা এখনই অনেক চিন্তিত। আমরা বেঁচে থাকি আর না থাকি, ASI তে পৌঁছালে সব কিছুই পরিবর্তন হয়ে যাবে। পৃথিবীর সকল শক্তি কিভাবে ব্যবহার করা যায়, আমরা তা বের করে ফেলতে পারব।  হয়তো পৃথিবী থেকে বের হয় অন্য গ্রহ গুলোতে সহজেই যেতে পারব। বিচরণ করতে পারব মহাবিশ্ব।

**আমরা কোন ধাপে রয়েছি?**

আমরা এখনো Artificial Narrow Intelligence ধাপে রয়েছি। আমরা যত রকম প্রোগ্রাম দেখি বা ব্যবহার করি, সব গুলোই ANI প্রোগ্রাম। Gogole Search Ingine, Self Driving Car, Flight Control, Nuclear Project Control and other complex system সব গুলোই ANI প্রোগ্রাম। এগুলো একটা নির্দিষ্ট কাজে দক্ষ। ঐ কাজের বাহিরে অন্য কোন কাজ করতে পারবে না।

**আমাদের অর্জনঃ**

১৯৯৭ সালে  IBM’s Deep Blue কম্পিউটার প্রোগ্রাম দাবার গ্র্যান্ডমাস্টারকে হারায়। যা একটি বুদ্ধিমান প্রোগ্রাম ছিল। ঐটা আর্টিফিশিয়াল ইন্টিলিজেন্স এর অনেক বড় একটা অর্জন ছিল।

এ বছর মার্চে AlphaGo প্রোগ্রাম গো বোর্ড গেমের ওয়ার্ল্ড চ্যাম্পিয়নকে হারায়। এটা প্রথম অর্জন থেকেও অনেক বড় একটা অর্জন। কারণ দাবা খেলাতে প্রতিটা পজিশন থেকে পরবর্তি মুভ দেওয়ার জন্য প্রায় ২৫টা পসিবল মুভ দেওয়া যায়। গো গেমে দেওয়া যায় ২০০ টা মুভ। কোন মুভ দিতে হবে, তা মানুষ যত সহজে চিন্তা করতে পারে, কম্পিউটারের জন্য এত সহজ না। এ ছাড়া দাবা খেলাটা হচ্ছে লজিক্যাল খেলা। লজিক দিয়েই খেলা যায়। কিন্তু গো খেলাটি বুদ্ধি দিয়ে খেলতে হয়। মানুষ যেভাবে চিন্তা করে, সে ভাবে চিন্তা করতে হয়।

**IBM Watson**

এছাড়া IBM Watson ২০১১ সালে অ্যামেরিকার বিখ্যাত কুইজ শো Jeopardy এর দুইজন চ্যাম্পিয়নকে হারায়।

IBM Watson হচ্ছে এখন পর্যন্ত সবচেয়ে সফল বুদ্ধিমান প্রোগ্রাম। এটা প্রথম দিকে ডেভেলপ করে প্রশ্ন করলে উত্তর দেওয়ার জন্য। Jeopardy এর দুই জন্ম চ্যাম্পিয়নকে হারিয়ে পথ চলা শুরু।  এখন নানা জায়গায় IBM ওয়াটসন ব্যবহার করা যায়। ডেটা এনালাইসিস, শিক্ষা, চিকিৎসা ইত্যাদি ভিন্ন ভিন্ন জায়গায় এই প্রোগ্রামটা ব্যবহার করা যায়।  ক্যান্সারের চিকিৎসায় চিকিৎসকদের ডিসিশন নিতে IBM Watson হেল্প করে। এছাড়া লাঞ্জ ক্যান্সারের নার্সদের প্রায় ৯০% IBM Watson এর দেওয়া ডিসিশনের উপর ভিত্তি করে কাজ করে। এই প্রোগ্রামের সবচেয়ে ভালো দিক হচ্ছে যত বেশি এটি ব্যবহার করা হবে, এ সিস্টেম তত উন্নত হতে থাকবে।

Watson API ব্যবহার করে যে কেউ ইন্টিজিলেজন্ট সফটওয়ার তৈরি করতে পারে। যদিও ফ্রি না!

**Personal Assistant:**

আমাদের কাছে পারসোনাল অ্যাসিস্ট্যান্ট সাইন্স ফিকশন গল্পের মত মনে হলেও এখন অনেক কাজই পারসোনাল এসিস্টেনট দিয়ে করা যায়। আপনি যে কোন কিছু ওকে জিজ্ঞেস করলে উত্তর দিবে। Ammazon Echo  বা Google Home এর মত প্রোডাক্ট গুলো দিয়ে কথা বলেই কম্পিউটারের সাথে যোগাযোগ করতে পারবেন। সাইন্স ফিকশন মুভি গুলোতে যেমন দেখা তেমন। আগে গল্প থাকলেও বিষয় গুলো এখন বাস্তব।

হিউম্যানয়েড রোবটঃ

মুভিগুলোতে মানুষের মত আমরা যে রোবট দেখি, সেগুলোকে বলে  হিউম্যানয়েড রোবট। এখন পর্যন্ত সেরা হিউম্যানয়েড রোবট হচ্ছে হোন্ডার ASIMO রোবট। এছাড়া বোস্টন ডায়নামিক মিলিটারি গ্রেড রোবট নিয়ে কাজ করছে। তাদের রোবট গুলো অনেক স্ট্রং। সবার জন্য উন্মুক্ত  হিউম্যানয়েড রোবট হচ্ছে NAO. এটি প্রোগ্রামেবল। আপনি এটিকে ক্রয় করে নিজের মত করে মডিফাই করে নিতে পারবেন। এটিতে IBM Watson এর মত প্রোগ্রাম সেট করলে গুগল হোম বা অ্যামাজন ইকো এর মত কাজ করবে।

**Self-Assemble Robot:**

সবচেয়ে ইন্টারেস্টিং রোবট হচ্ছে সেলফ এসেম্বলল রোবট। যারা বিগ হিরো মুভিটি দেখেছেন, সেখানে মাইক্রোবট দেখানো হয়। ছোট ছোট রোবট। যেগুলো এক সাথ হয়ে যে কোন আকার ধারণ করতে পারে। কয়েক দিন আগে মুভি দেখলে এখন দেখি বাস্তবে এরকম রোবট নিয়ে কাজ হচ্ছে। এবং সফল ও হয়েছে। MIT এর Self Assembly Lab এ রকম রোবট নিয়ে কাজ হচ্ছে। এবং দারুণ সব মাইক্রোবট তৈরি করেছে।

ন্যানো টেকনোলজি যখন অনেক অগ্রসর হবে, তখন এই সেলফ এসেম্বেল রোবট গুলো দিয়ে মানুষের শরীরের যে কোন রোগ সারানো সম্ভব হবে। শরীরে ভাইরাস? মাইক্রোবট রক্তকণিকায় প্রবেশ করে ভাইরাস গুলো ধ্বংস করবে। ক্যান্সারের মত রোগ বলতে কিচ্ছু থাকবে না।

**Industrial Robot.**

এমন না যে রোবট গুলো দেখতে মানুষের মত হবে। রোবট যে কোন রকম হতে পারে। ইন্ড্রাস্টিতে অনেক কাজই রোবট দিয়ে করা হয়। গাড়ি তৈরি থেকে শুরু করে মোবাইলের চিপ বা প্রসেসর তৈরির মত  কাজ গুলো রোবট দিয়ে করা হয়।

**Machine Learning**

আমরা শুধু ANI ব্যবহার করেই অনেক গুলো সমস্যা সমাধান করে ফেলছি। আমাদের জীবনকে অনেক সহজ করে তুলেছি। এই ANI প্রোগ্রাম গুলো তৈরি করার সময় আমরা সরাসরি  বলে দেই না কি করতে হবে। আমরা কিছু উদাহরণ দেই। এরপর ঐ উদাহরণ থেকেই প্রোগ্রামটা একটা অ্যালগরিদম দাড় করে ফেলে। এরপর পরবর্তিতে  যে কাজ করতে বলা হয়, নিজের মত করে করে। এটাই হচ্ছে মেশিন লার্নিং।

কিছুক্ষণ আগে যত গুলো প্রোগ্রাম বা রোবটের কথা বলেছি, সব গুলোতেই মেশিন লার্নিং এর ব্যবহার রয়েছে। আগামী দুই তিন বছরের মত যত সফটওয়ার বা অ্যাপ রয়েছে, সব গুলোতে মেশিন লার্নিং ব্যবহার করা হবে।

**Prisma**

আমরা প্রিসমা অ্যাপ অনেকেই ব্যবহার  করেছি। যারা ব্যবহার করি নি, তারা অন্তত প্রিসমা দিয়ে তৈরি করা ছবি দেখেছি। সাধারণ ছবিকে কি সুন্দর আর্টিস্টিক ছবি তৈরি করে দেয়। এই অ্যাপটির মূলে রয়েছে ডীপ নিউরাল নেটওয়ার্ক। যা মেশিন লার্নিং এর একটা পদ্ধতি। এখন দারুণ কিছু করতে মেশিন লার্নিং প্রয়োগ করতেই হবে আমাদের।

**মেশিন লার্নিং এর জব মার্কেটঃ**

আপ ওয়ার্ক প্রতি চার মাস পর পর স্কিল ইনডেক্স শেয়ার করে। এখন সবচেয়ে গ্রোইং স্কিল হচ্ছে মেশিন লার্নিং! শুধু ভবিষ্যৎ না, বর্তমানেও মেশিন লার্নিং এর অনেক চাহিদা রয়েছে।

**শিখব কোথায় থেকে?**

অনলাইনে প্রচুর রিসোর্স রয়েছে। আমার প্রিয় সাইট হচ্ছে ইউডাসিটি।  ফ্রি রিসোর্সের পাশাপাশি প্রিমিয়াম রিসোর্স ও রয়েছে। গিট হাবে [**Awesome Artificial Intelligence**](https://github.com/owainlewis/awesome-artificial-intelligence) এবং [**Awesome Machine Learning**](https://github.com/josephmisiti/awesome-machine-learning/blob/master/books.md)নামে দুইটা প্রজেক্ট রয়েছে। যেখানে আর্টিফিশিয়াল ইন্টিলিজেন্স এবং মেশিন লার্নিং এর উপর যত রিসোর্স রয়েছে, সব গুলোর লিস্ট রয়েছে। যা নিজের কাছে সহজ মনে হয়, সেখান থেকেই শুরু করতে পারেন।

### পাইথন প্রোগ্রামিং ল্যাঙ্গুয়েজ ব্যবহার করে প্রথম মেশিন লার্নিং প্রোগ্রাম

মেশিন লার্নিং নিয়ে কাজ করা কঠিন কিছু না। আমরা ছোট্ট একটা মেশিন লার্নিং প্রোগ্রাম লিখব। ৫ লাইনের মত মাত্র। নিজেরাই দেখব, কত সহজ একটা মেশিন লার্নিং প্রোগ্রাম লেখা যায়। প্রোগ্রামটা লিখব আমরা পাইথন প্রোগ্রামিং ল্যাঙ্গুয়েজ ব্যবহার করে। সাথে ব্যবহার করব [**scikit-learn**](http://scikit-learn.org/)।

scikit-learn হচ্ছে পাইথনের মেশিন লার্নিং লাইব্রেরী। [**পাইথন**](https://jakir.me/python) নিয়ে এই ব্লগে অনেক লেখা রয়েছে। পাইথনের ব্যাসিক ধারণা নেওয়ার জন্য সেগুলো দেখতে পারেন।

পাইথন যদি ইন্সটল করা না থাকে, তাহলে পাইথন ইন্সটল করে নিতে হবে। এরপর আমরা scikit-learn ইন্সটল করব। অনেক ভাবেই ইন্সটল করা যায়। PyPy এর সাহায্য ইন্সটল করতে টার্মিনাল বা কমান্ডলাইনে গিয়ে নিচের কমান্ডটা লিখলেই হবেঃ

|  |  |
| --- | --- |
| 1 | pip install -U scikit-learn |

PyPy হচ্ছে প্যাকেজ ম্যানেজার। PyPy কিভাবে ইন্সটল করতে হয়, তা সম্পর্কে জানা যাবে পাইথনের মডিউল এবং প্যাকেজ নামক লেখাটিতে। http://jakir.me/modules-package

নরমলা যে কোন টেক্সট এডিটরেই আমরা কোড লিখতে পারি। অথবা যে কোন পাইথন IDE ব্যবহার করতে পারি। পাইথনের জন্য আমার প্রিয় IDE হচ্ছে [**PyCharm**](https://www.jetbrains.com/pycharm/download/). এখানে গিয়ে PyCharm এর কমিউনিটি এডিশনটা ডাউনলোড করে নেওয়া যাবে।

আমরা চাইলে [**Anaconda**](https://www.continuum.io/downloads) ও ব্যবহার করেও।  scikit-learn ইন্সটল করতে পারি। তার জন্য প্রথমে Anaconda ইন্সটল করতে হবে। এরপর কমান্ড লাইন অথবা টার্মিনালে গিয়ে লিখতে হবেঃ

|  |  |
| --- | --- |
| 1 | conda install scikit-learn |

এতটুকু করলে আমরা প্রথম মেশিন লার্নিং প্রোগ্রাম লেখার জন্য প্রস্তুত। মেশিন লার্নিং সম্পর্কে ধারণা না থাকলে এই লেখা দুইটি দেখতে পারেনঃ

* [**মেশিন লার্নিং সম্পর্কে ধারণা**](https://jakir.me/%E0%A6%AE%E0%A7%87%E0%A6%B6%E0%A6%BF%E0%A6%A8-%E0%A6%B2%E0%A6%BE%E0%A6%B0%E0%A7%8D%E0%A6%A8%E0%A6%BF%E0%A6%82)
* [**মেশিন লার্নিং এর কিছু টার্ম**](https://jakir.me/%E0%A6%AE%E0%A7%87%E0%A6%B6%E0%A6%BF%E0%A6%A8-%E0%A6%B2%E0%A6%BE%E0%A6%B0%E0%A7%8D%E0%A6%A8%E0%A6%BF%E0%A6%82-%E0%A6%8F%E0%A6%B0-%E0%A6%9F%E0%A6%BE%E0%A6%B0%E0%A7%8D%E0%A6%AE)

আপনাকে এমন একটা প্রোগ্রাম লিখতে বলল, যেখানে একটা ছবি দিয়ে বলা হলো এটা ছেলের ছবি না কি মেয়ের ছবি, তা বের করে দিতে হবে। তখন আপনি কি করবেন? যদি আমরা এ প্রোগ্রাম লেখার জন্য ম্যানুয়াল রুল লিখি, তাহলে সিম্পল এ কাজের জন্য কয়েক হাজার কোড লিখতে হতে পারে। আবার একটা মেয়ে যদি ছেলের মত করে চুল কাটে, তাহলে প্রোগ্রামটি ভুল রেজাল্ট দিতে পারে। ছেলে যদি বড় চুল রাখে, তাহলেও হয়তো প্রোগ্রামটি ভুল রেজাল্ট দিবে। এরকম প্রতিটা পরিস্থিতির কথা মাথায় রেখে কোড লিখতে হবে। যত প্রিসাইস করেই আমরা কোড লিখি না কেন, দেখা যাবে কোন না কোন পরিস্থিতিতে আমাদের প্রোগ্রাম ভুল রেজাল্ট দিচ্ছে।

এরকম সমস্যা সমাধানের জন্য আমাদের এমন অ্যালগরিদম ডিজাইন করা দরকার, যে আমাদের হয়ে প্রতিটা পরিস্থিতির কথা চিন্তা করে রুল বা লজিক গুলো সাজাবে। আমরা শুধু আমাদের অ্যালগরিদমকে কিছু ডেটা দিয়ে ট্রেইন করে নিব।

মানুষের একটা ছবি আমরা ইনপুট হিসেবে দিব, আউটপুট হিসেবে আমাদের দিবে ঐটা ছেলে নাকি মেয়ের ছবি। ছবিকে আমরা ক্লাসিফাই করছি। আলাদা আলাদা ক্লাসে সাজাচ্ছি। বিষয়টাকে আমরা ফাংশনের সাথে তুলনা করতে পারি। আর এই ফাংশনটিকে বলা হয় ক্লাসিফিয়ার। আর অটোমেটিকেলি কোন ক্লাসিফায়ার লেখাকে বলা হয় Supervised Learning।

**Supervised Learning** লার্নিং এ আমরা কিছু উদাহরণ বা ট্রেইনিং ডেটা দিয়ে দেই। ট্রেনিং ডেটা দিয়ে আমাদের ক্লাসিফায়ারকে ট্রেইন করি। যেমন আমরা বলি কোন ছবিতে যদি মানুষটির চুল বড় থাকে এবং ওজন কম হয়, তাহলে তা মেয়ে হবে। এটা শুধু একটা ডেটা। এমন আরো অনেক গুলো ডেটা দেওয়া থাকে। তখন ক্লাসিফায়ারটা ঐ ডেটা গুলো দেখে নিজে নিজে চিন্তা করতে পারে। এরপর আমরা যদি একটা ছবি দিয়ে বলি, বল তো এই ছবিটা কি মেয়ের ছবি নাকি ছেলের ছবি, তখন সে সুন্দর করে বলে দেয়। আমরা এখন সিম্পল একটা প্রোগ্রাম লিখব।

তার জন্য আমরা কিছু ট্রেইনিং ডেটা সংগ্রহ করব। ডেটার দুইটা অংশ থাকে, ফিচার এবং লেভেল। আমাদের সিম্পল প্রোগ্রাম লেখার জন্য আমরা এই ডেটা গুলো ব্যবহার করতে পারিঃ

|  |  |  |
| --- | --- | --- |
| Feature 1: Hair | Feature 2: Weight | Label |
| Long | 50 | Female |
| Short | 60 | Male |
| Long | 45 | Female |
| Short | 70 | Male |

একটা মানুষের বিভিন্ন বৈশিষ্ট্য গুল হচ্ছে ফিচার। আর তা ছেলে, নাকি মেয়ে তাকে আমরা বলি লেভেল। একটা অবজেক্টের অনেক গুলো ফিচার থাকতে পারে। আমরা সিমপ্লিসিটির জন্য দুইটা ফিচার নিয়ে কাজ করব। Hair Size এবং ওজন। যত বেশি ট্রেনিং ডেটা থাকবে, ক্লাসিফায়ার তত ভালো কাজ করবে।

ক্লাসিফায়ার হচ্ছে অনেক গুলো রুল এর কালেকশন। আমরা আমাদের ক্লাসিফায়ারে ব্যবহার করব Decision Tree. নাম থেকেই বোঝা যাচ্ছে Decision Tree কি জিনিস।  ডিসিশন ট্রি ছাড়াও আরো অনেক ধরনের ক্লাসিফায়ার রয়েছে। যেমন **Artifical Nural Nework**, **Support Vector Machine** ইত্যাদি।

সিম্পল প্রোগ্রাম লেখার জন্য আমরা long = 1 ধরব এবং short = 0 ধরব।

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | from sklearn import tree  features = [[1, 50], [0, 60], [1, 45], [0, 70]]  labels = ['Female', 'Male', 'Female', 'Male']    clf = tree.DecisionTreeClassifier()  clf = clf.fit(features, labels)  print (clf.predict([[1,55]])) |

দুইটা ভ্যারিয়েবল নিয়েছি আমরা। একটাতে রেখেছি ফিচার গুলো, আরেকটাতে লেভেল গুলো।

ফিচার হচ্ছে আমাদের ইনপুট, আর লেভেল হচ্ছে আমাদের আউটপুট। প্রোগ্রামকে আমরা একটা মানুষের ফিচার দিয়ে বলব তা ছেলে না মেয়ে, তা বলে দিতে। আর তা করার জন্য আমরা ডিসিশন ট্রি ব্যবহার করব। sikit learn এ ডিসিশন ট্রি অ্যালগরিদম ইমপ্লিমেন্ট করা রয়েছে। আমরা তা ব্যবহার করব। তার জন্য আমাদের প্রথমে আমরা DecisionTreeClassifier টা ইনিশিয়ালাইজ করে নিব এবং এর মধ্যে আমাদের ট্রেনিং ডেটা পাস করব।

ট্রেনিং ডেটা থেকে ডিসিশন ট্রি এর মাধ্যমে প্রোগ্রামটা একটা লজিক দাড় করিয়ে নিবে নিজে নিজে। এরপর যদি আমরা কোন মানুষের চুলের সাইজ এবং ওজন ইনপুট হিসেবে দেই, প্রোগ্রামটা আমাদের বলে দিবে সে ছেলে নাকি মেয়ে।

Yes! আমরা আমাদের প্রথম মেশিন লার্নিং প্রোগ্রাম তৈরি করেছি। এরপর? নিজে একটা সমস্যা বের করে মেশিন লার্নিং এর মাধ্যমে সমাধান করার চেষ্টা করতে পারেন।

### আর্টিফিশিয়াল ইন্টেলিজেন্স বা কৃত্রিম বুদ্ধিমত্তা সম্পর্কে ধারণা এবং আমাদের অবস্থান

কম্পিউটার বা যে কোন মেশিন হচ্ছে বোকা বাক্স। এগুলোকে কাজে লাগানোর জন্য ইন্ট্রাকশনের দরকার হয়। প্রোগ্রামিং ল্যাঙ্গুয়েজ ব্যবহার করে আমরা কম্পিউটার বা যে কোন মেশিনকে কিছু ইন্সট্রাকশন দেই, কম্পিউটার বা মেশিন গুলো সে অনুযায়ী কাজ করে।

এই মেশিনকে আমরা যে ইন্সট্রাকশনই দিব, মেশিন সে অনুযায়ীই কাজ করবে। এর বাহিরে নিজ থেকে কিছু করতে পারবে না। মেশিন যেন নিজ থেকে কিছু করতে হলে তার কিছু বুদ্ধি শুদ্ধি লাগবে। মেশিনের বুদ্ধি শুদ্ধিকে আমরা বলি আর্টিফিশিয়াল ইন্টেলিজেন্স বা কৃত্রিম বুদ্ধিমত্তা। একটা রোবটের কথা যদি চিন্তা করি, রোবটের বুদ্ধি হচ্ছে আর্টিফিশিয়াল ইন্টেলিজেন্স।

বুদ্ধি জিনিসটা কি?

বুদ্ধি হচ্ছে জ্ঞান আহরণ করা এবং তা প্রয়োগ করার ক্ষমতা। সাধারণ প্রোগ্রাম গুলো জ্ঞান আহরণ করতে পারে না। কিন্তু যে সব মেশিন বা প্রোগ্রাম এমন ভাবে তৈরি করা হয়, যেন নিজে নিজে কিছু শিখে নিতে পারে, সেগুলোকে আমরা বলি বুদ্ধিমান প্রোগ্রাম বা বুদ্ধিমান মেশিন। যেমন গুগল সার্চ প্রোগ্রামটা একটা বুদ্ধিমান প্রোগ্রাম। আমরা কিছু সার্চ করলে এটি আমাদের আগের সার্চ হিস্টোরি, বয়স, লোকেশন ইত্যাদির উপর নির্ভর করে আমাদের সার্চ রেজাল্ট দেখায়।

আর্টিফিশিয়াল ইন্টেলিজেন্স এর লক্ষ্য হচ্ছে কম্পিউটার বা মেশিনকে মানুষের মত জ্ঞান দান করা। মানুষের মত চিন্তা করার ক্ষমতা দান করা।।

আর্টিফিশিয়াল ইন্টেলিজেন্স যদি আমরা খুব উন্নত করতে পারি, হয় এটি হবে সবচেয়ে দারুণ একটা পরিবর্তন অথবা সবচেয়ে ভয়ঙ্কর পরিবর্তন। এমনকি আমাদের অস্তিত্বও ধ্বংস হয়ে যেতে পারে। আমরা মানুষেরা বুদ্ধিমান, কিউরিয়াস। আমরা দেখতে চাই কি হবে ফিউচারে। এটাই হচ্ছে বুদ্ধি। আর এ জন্যই আমরা আর্টিফিশিয়াল ইন্টেলিজেন্স এর ডেভেলপ করে যাবো। আমরা চেষ্টা করে যাবো কম্পিউটারকে মানুষের মত বুদ্ধিমান করে তোলার।

বিভিন্ন এক্সপার্ট এর মতে আর্টিফিশিয়াল ইন্টেলিজেন্স বা কৃত্রিম বুদ্ধিমত্তার তিনটা ক্যাটেগরি রয়েছে।

**ANI বা Artificial Narrow Intelligence:** ANI হচ্ছে একটা নির্দিষ্ট এরিয়াতে এক্সপার্ট। যেমন যে মেশিন দাবা খেলতে পারবে, সে শুধু দাবাই ভালো পারবে। দাবা খেলা থেকে লুডু খেলা সহজ হওয়া সত্ত্বেও তাকে দাবা খেলার পরিবর্তে যদি লুডু খেলতে দেয়, সে পারবে না। ANI কে Weak AI ও বলা হয়। একে আমরা বলতে পারি কৃত্রিম বুদ্ধিমত্তার প্রথম ধাপ।

**AGI বা Artificial General Intelligence:** কৃত্রিম বুদ্ধিমত্তার দ্বিতীয় ধাপ হচ্ছে AGI। এটাকে Strong AI, বা Human-Level AI ও বলা হয়। এ ধাপে কম্পিউটার মানুষের মত চিন্তা করতে পারবে, মানুষের মত প্ল্যান করা, সমস্যা সমাধান করা, হঠাৎ নতুন কোন পরিবেশে আসলে চারপাশ দেখে সে পরিবেশের সাথে মানিয়ে নেওয়ার মত ক্ষমতা অর্জা করবে।

**ASI বা Artificial Super Intelligence:** কৃত্রিম বুদ্ধিমত্তার তৃতীয় ধাপ। কম্পিউটার যখন মানুষ থেকেও বুদ্ধিমান হবে তখন তাকে আমরা বলব আর্টিফিশিয়াল সুপার ইন্টেলিজেন্স। আর্টিফিশিয়াল সুপার ইন্টেলিজেন্স সিস্টেম তৈরি হলে কি হবে, মানুষের ভালো হবে না খারাপ হবে, তা নিয়ে গবেষকরা এখনই চিন্তিত।

যদিও এখনো চিন্তা করার মত কিছু হয় নি আমরা এখনো ANI তে পড়ে রয়েছি। যত বুদ্ধিমান মেশিন বা প্রোগ্রাম রয়ছে, সব গুলোই একের অধিক আর্টিফিশিয়াল ন্যারো ইন্টেলিজেন্স এর সমন্বয়ে গঠিত।

আমাদের হাতের স্মার্টফোনটিতে অনেক গুলো ANI প্রোগ্রাম রয়েছে। ফোনের সবচেয়ে সফল ANI প্রোগ্রাম হচ্ছে সিরি বা কর্টনা। এছাড়া গুগলের Allo হচ্ছে ANI এর একটা চমৎকার প্রয়োগ। Allo এখনো ব্যবহার করে না থাকলে একটু ইন্সটল করে ব্যবহার করে দেখুন। গুগল এসিস্টেন্ট এর সাথে চ্যাট করুন। বুঝতে পারবেন AI কতটুকু উন্নত হয়েছে। গুগলের সেলফ ড্রাইভিং কার ANI এর সফল প্রয়োগের উদাহরণ। ফেসবুক নিজেও ANI ফ্যাক্টরি বলা যায়। অ্যামাজন বা সব বড় বড় ওয়েব সাইটেই ANI এর ব্যবহার রয়েছে। প্রতিটা প্লাইট ANI সিস্ট্রেম দ্বারা পরিচালিত। নিউক্লিয়ার প্ল্যান্ট গুলো পরিচালনা করতে ANI সিস্টেম ব্যবহার করা হয়। ANI সর্বোচ্চ যে ক্ষতিটি করতে পারে তা হচ্ছে প্লেন ক্র্যাস, নিউক্লিয়ার প্ল্যান্ট ক্র্যাশ, গুগল সেলফ ড্রাইভিং কার সহজলভ্য হলে ছোট খাটো এক্সিডেন্ট ইত্যাদি।

[**এলান টিউরিংকে**](https://jakir.me/%E0%A6%8F%E0%A6%B2%E0%A6%BE%E0%A6%A8-%E0%A6%9F%E0%A6%BF%E0%A6%89%E0%A6%B0%E0%A6%BF%E0%A6%82-%E0%A6%B8%E0%A6%AE%E0%A7%9F-%E0%A6%A5%E0%A7%87%E0%A6%95%E0%A7%87-%E0%A6%8F%E0%A6%97%E0%A6%BF%E0%A7%9F) বলা হয় কৃত্রিম বুদ্ধিমত্তার জনক। ১৯৫০ সালের দিকে এলান টিউরিং একটি মেশিন বুদ্ধিমান কিনা, তা পরীক্ষা করার জন্য একটি টেস্ট এর কথা উল্যেখ করে গিয়েছেন, যা টিউরিং টেস্ট নামে পরিচিত। ঐ সময় কৃত্রিম বুদ্ধিমত্তা নিয়ে অনেক রিসার্চ হলেও এর পর অনেক দিন AI নিয়ে রিসার্চ বন্ধ থাকে। প্রধান একটা কারণ হিসেবে ধরা হয় কম্পিউটেশনাল পাওয়ার। ঐ সময়কার কম্পিউটার গুলো এত পাওয়ারফুল ছিল না। কম্পিউটারের প্রসেসিং পাওয়ার বাড়ার সাথে সাথে কৃত্রিম বুদ্ধিমত্তা নিয়ে আবার রিসার্চ শুরু হয়েছে। কিছুদিন আগে Facebook, Google, Amazon মিলে AI এর উপর রিসার্চ করার জন্য চুক্তিবদ্ধ হয়েছে। Elon Musk গঠন করেছেন OpenAI নামক প্লাটফরম।

বুদ্ধিমান প্রোগ্রাম তৈরি করার জন্য প্রধান যে বিষয়টা দরকার তা হচ্ছে Knowledge Representation & Reasoning। আর সুন্দর ভাবে Knowledge Representation & Reasoning এর জন্য মানুষের ব্রেইন কিভাবে কাজ করে, বিজ্ঞানীরা তার অনুকরণ করার চেষ্টা করছে।

আমরা মহাকাশ নিয়ে গবেষণা করি। মঙ্গলে মানুষের কলোনি তৈরি করার স্বপ্ন দেখি। সবই করি এই মস্তিষ্কটাকে কাজে লাগিয়ে। মহাকাশ অনেক দূরে হয়েও আমরা অনেক কিছু জানতে পারি। কিন্তু মস্তিষ্কটা কিভাবে কাজ করে, এখনো ঠিক মত আমরা বুঝে উঠতে পারি নি। যতটুকু বুঝে উঠেছি আমরা, তত টুকু ব্যবহার করেই আমরা বুদ্ধিমান সিস্টেম তৈরি করার চেষ্টা করছি।

মানুষের মস্তিক নিউরন দ্বারা গঠিত। আমাদের মস্তিষ্কে প্রায় ১০০ বিলিয়ন নিউরন রয়েছে। এগুলো একটা একটার সাথে কানেক্টেড, একটা নেটওয়ার্ক এর মত। এই জৈবিক নিউরাল নেটওয়ার্ক এর অনুকরণ করেই [**আর্টিফিশিয়াল নিউরাল নেটওয়ার্ক**](https://jakir.me/%e0%a6%a8%e0%a6%bf%e0%a6%89%e0%a6%b0%e0%a6%be%e0%a6%b2-%e0%a6%a8%e0%a7%87%e0%a6%9f%e0%a6%93%e0%a7%9f%e0%a6%be%e0%a6%b0%e0%a7%8d%e0%a6%95) তৈরি করা হয়েছে।

আর্টিফিশিয়াল নিউরাল নেটোওয়ার্ক কৃত্রিম বুদ্ধিমত্তার একটা অংশ মাত্র। মূলত এটি হচ্ছে [**মেশিন লার্নিং**](https://jakir.me/%e0%a6%ae%e0%a7%87%e0%a6%b6%e0%a6%bf%e0%a6%a8-%e0%a6%b2%e0%a6%be%e0%a6%b0%e0%a7%8d%e0%a6%a8%e0%a6%bf%e0%a6%82) এর একটা শাখা। আর মেশিন লার্নিং হচ্ছে আর্টিফিশিয়াল ইন্টেলিজেন্স এর একটি শাখা। মেশিন লার্নিং ছাড়া AI এর অন্যান্য বিষয় গুলো হচ্ছে ন্যাচারাল ল্যাঙ্গুয়েজ প্রসেসিং, অবজেক্ট রিকগনিশন, প্যাটার্ন রিকগনিশন, রোবটিক্স, ইভোলিউশনারি কম্পিউটেশন যেমন জেনেটিক অ্যালগরিদম, [**ফাজি সিস্টেম**](https://jakir.me/%e0%a6%ab%e0%a6%be%e0%a6%9c%e0%a6%bf-%e0%a6%b2%e0%a6%9c%e0%a6%bf%e0%a6%95), প্রবাবিলিটি, প্রিডিকশন, Knowledge management সহ অন্যান্য।

ANI সিস্টেম ১৯৯৭ সালে মানুষকে হারিয়ে দিয়েছে। [**ডীপ ব্লু**](https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer)) নামের একটি কম্পিউটার হারিয়ে দেয় বিশ্বখ্যাত দাবার গ্রান্ডমাস্টার চ্যাম্পিয়ন গ্যারি কাসপারভকে। ডিপ ব্লু-এর পর এ বছর মার্চে [**আলফাগো**](https://deepmind.com/research/alphago/) মানুষকে হারায়। আলফাগো নামক কম্পিউটার প্রোগ্রামটি তৈরি করে গুগলের ডীপমাইন্ড নামক প্রতিষ্ঠান। এই প্রোগ্রামটি তৈরি করা হয় ‘গো’ গেমটি খেলার জন্য। গো গেমটি একটি প্রাচীন চাইনিজ গেম।

স্মার্ট হোম, স্মার্ট কার, ভার্চুয়াল অ্যাসিস্ট্যান্ট, রোবট সহ নানা জায়গায় কৃত্রিম বুদ্ধিমত্তা ব্যবহার করে আমরা আমাদের জীবনকে সহজ করতে পারি, পারব। বিশেষ করে মেডিক্যাল সাইন্স এবং কৃষি কাজে কৃত্রিম বুদ্ধিমত্তা ব্যবহার করে পৃথিবী থেকে রোগ এবং দারিদ্র্যতা দূর করা সম্ভব। আমরা যারা দারুণ কিছু শিখতে চাই, তারা আর্টিফিশিয়াল ইন্টেলিজেন্স নিয়ে পড়া লেখা করতে পারি। জ্ঞানের ক্ষুদা আর ক্যারিয়ার, দুইটা এক সাথেই মিটবে আশা করি।

### কৃত্রিম বুদ্ধিমত্তা সম্পর্কে শিখতে চাইলেঃ

* Udacity এর [**Intro to Artificial Intelligence**](https://www.udacity.com/course/intro-to-artificial-intelligence--cs271) কোর্সটা প্রাথমিক ধারণা নেওয়ার জন্য খুবি দারুণ। সব গুলো ইউনিভার্সিটিতে Artificial Intelligence: A Modern Approach বইটি থেকে পড়ানো হয়। নীলক্ষেত থেকে বইটি কিনতে পাওয়া যাবে। আর বইটির সহ লেখক হচ্ছেন Peter Norvig। পিটার নরভিগ হচ্ছেন গুগলের রিসার্চ টিমের ডিরেক্টর। আর Udacity’র এ কোর্সটার সহ ইন্সট্রাকটর। পিটার নরভিগের একটা দারুণ লেখা রয়েছে। [**Teach Yourself Programming in Ten Years**](http://norvig.com/21-days.html)। যারা পড়েন নি, একবার পড়ে নিতে বলব। এছাড়া এই কোর্সটি করার পর ইউডাসিটিতে আরো কিছু কোর্স রয়েছে, যেমন মেশিন লার্নিং ইত্যাদি। সেগুলোও দেখতে পারেন।
* edX এর [**Artificial Intelligence**](https://www..org/course/artificial-intelligence-uc-berkeleyx-cs188-1x)কোর্সটাও দেখতে পারেন। ব্যাসিক গুলো জানার পর অনলাইনে অনেক রিসোর্স রয়েছে শেখার মত। সাহস করে শুরু করলেই হয়।
* [**Stanford এর AI কোর্সের আউটলাইন, স্লাইড গুলো পাওয়া যাবে এখানে।**](http://web.stanford.edu/class/cs221/)
* [**প্রচুর ধৈর্য্য থাকলে MIT এর তৈরি করা ভিডিও গুলো দেখতে পারেন।**](https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/)এ ছাড়া লেকচার স্লাইড গুলোও ডাউনলোড করে পড়া যাবে।  শেখা শুরু করার জন্য এর থেকে বেশি কিছু আশা করি লাগবে না।

### আর্টিফিশিয়াল ইন্টিলিজেন্স নিয়ে পড়ার পাশা পাশি বিনোদনের জন্য এগুলো দেখতে পারেনঃ

**Movies:**

* Ex Machina (2015)
* 2001: A Space Odyssey
* I, Robot
* Matrix series
* Chappie
* Transcendence
* The Terminator series
* Star Trek series
* Tron series
* Her (2013)
* A.I. Artificial Intelligence ইত্যাদি।

**TV series:**

* Person of Interest
* Human
* Intelligence
* Mr Robot ইত্যাদি।

## পাইথন ডেটাবেজঃ SQLite এবং পাইথন

জনপ্রিয় প্রায় যে কোন ডেটাবেজ ম্যানেজমেন্ট সিস্টেমই আমরা পাইথনের সাথে ব্যবহার করতে পারি। সবচেয়ে সিম্পল ডেটাবেজ ম্যানেজমেন্ট সিস্টেম হচ্ছে SQLite। একটা সম্পর্কে ভালো করে জানলে বাকি সব গুলো ডেটাবেজ একই ভাবেই ব্যবহার করা যাবে। মূল বিষয় গুলো একই। একটার সাথে একটার কিছু পার্থক্য আছে, কিছু সুবিধে রয়েছে। আমাদের প্রজেক্ট অনুযায়ী আমরা ডেটাবেজ সিলেক্ট করতে পারব। শুরু করি SQLite দিয়ে। অন্য ডেটাবেজ গুলো ব্যবহার করার জন্য আমাদের আলাদা ভাবে সে গুলো ইন্সটল করতে হয়। কিন্তু পাইথনের সাথে SQLite ইন্সটল করা থাকে। আমাদের বাড়তি কোন কফিগারেশন করতে হবে না।

SQLite ব্যবহার করার জন্য প্রথমে আমাদের ইম্পোর্ট করে নিতে হবে। আমরা যে ভার্সনটা ব্যবহার করব তা হচ্ছে 3, তাই ইম্পোর্ট করার জন্য আমাদের লিখতে হবেঃ

|  |  |
| --- | --- |
| 1 | import sqlite3 |

ডাটাবেজ তৈরি করা ভয়াবহ রকম সোজা। তার জন্য লিখতে হবেঃ

|  |  |
| --- | --- |
| 1  2  3 | import sqlite3    db = sqlite3.connect('friends.db') |

এখানে friends.db হচ্ছে ডেটাবেজ নেম। যে নামে আমরা ডেটাবেজ তৈরি করব।

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | import sqlite3    db = sqlite3.connect('friends.db')    db.execute('drop table if exists friends')    db.execute('create table friends (i int, friend text)')    db.execute('insert into friends(i, friend) values (?,?)', ( 1, 'Shuvo ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 2, 'Tuhin ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 3, 'Tanvir ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 4, 'Dipu ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 5, 'Siyam ' ))  db.commit()    result = db.execute('select \* from friends')  for row in result :      print (row) |

প্রোগ্রামের শুরুতেই আমরা দেখে নিয়েছি friends নামে কোন টেবিল আছে কিনা, যদি থাকে, তাহলে আমরা তা রিমুভ করব drop কমান্ড দিয়ে। কারণ আমরা এরপর friends নামে একটা টেবিল তৈরি করব।

|  |  |
| --- | --- |
| 1 | db.execute('drop table if exists friends') |

টেবিল তৈরি করার কমান্ডঃ

|  |  |
| --- | --- |
| 1 | db.execute('create table friends (i int, friend text)') |

যেখানে আমরা দুইটা কলাম তৈরি কব। একটা হচ্ছে i নামে। যেটায় ইন্টিজার ভ্যালু রাখব। আরেকটা হচ্ছে friend নামে, যেটায় আমরা টেক্সট রাখব।

এবার আমরা ডেটাবেজে কয়েকটি ডেটা রাখব, তার জন্যঃ

|  |  |
| --- | --- |
| 1 | db.execute('insert into friends(i, friend) values (?,?)', ( 1, 'Shuvo ' )) |

insert into friends(i, friend) এর friends হচ্ছে ডেটাবেজ টেবিলের নাম। i, friend হচ্ছে দুটি কলাম, যেখানে আমরা ডেটা রাখব। values (?,?) এখানে ? হচ্ছে প্লেসহোল্ডার। ( 1, ‘Shubo ‘ ) হচ্ছে ভ্যালু। প্রথম প্লেস হোল্ডারের নাম হচ্ছে 1, দ্বিতীয় প্লেসহোল্ডারের মান হচ্ছে Shuvo.

এভাবে আমরা কয়েকটি মান রেখেছি।

এর পরের কাজ হচ্ছে ডেটা বেজ থেকে ডেটা read করা বা পড়া।

তার জন্য db.execute(‘select \* from friends’) দিয়ে friends টেবিলের সব ডেটা আমরা পড়ে result নামক ভ্যারিয়েবলে রেখেছি। result ভ্যারিয়েবলে ডেটা গুলো টাপল অনুযায়ী রয়েছে। আমরা এবার তা প্রিন্ট করতে পারিঃ

|  |  |
| --- | --- |
| 1  2 | for row in result :       print (row) |

ডেটা গুলো অর্ডার করতে যে রো দিয়ে অর্ডার করতে চাই, তা উল্লেখ করে দিতে পারি, তার জন্য SQL কমান্ড হচ্ছেঃ select \* from friends order by i

সম্পুর্ণ প্রোগ্রামঃ

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | import sqlite3    db = sqlite3.connect('friends.db')    db.execute('drop table if exists friends')    db.execute('create table friends (i int, friend text)')    db.execute('insert into friends(i, friend) values (?,?)', ( 1, 'Shubo ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 2, 'Tuhin ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 3, 'Tanvir ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 4, 'Dipu ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 5, 'Siyam ' ))  db.commit()    result = db.execute('select \* from friends order by i')  for row in result :      print (row) |

আমরা চাইলে ফ্রেন্ড দিয়েও অর্ডার করতে পারিঃ

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | import sqlite3    db = sqlite3.connect('friends.db')    db.execute('drop table if exists friends')    db.execute('create table friends (i int, friend text)')    db.execute('insert into friends(i, friend) values (?,?)', ( 1, 'Shubo ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 2, 'Tuhin ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 3, 'Tanvir ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 4, 'Dipu ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 5, 'Siyam ' ))  db.commit()    result = db.execute('select \* from friends order by friend')  for row in result :      print (row) |

ডেটা বেজে অনেক ডেটা থাকবে। উপরের প্রোগ্রামে আমরা friends টেবিল থেকে সব গুলো ডেটা রিড করেছি। আমরা চাইলে শুধু i কলাম বা শুধু friend কলাম থেকে ডেটা রিড করতে পারি। যেমন আমরা শুধু friend কলামের ডেটা গুলো রিড করব, তার SQL কমান্ড হছেঃ select friend from friends order by friend

সম্পুর্ণ প্রোগ্রামঃ

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | import sqlite3    db = sqlite3.connect('friends.db')    db.execute('drop table if exists friends')    db.execute('create table friends (i int, friend text)')    db.execute('insert into friends(i, friend) values (?,?)', ( 1, 'Shubo ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 2, 'Tuhin ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 3, 'Tanvir ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 4, 'Dipu ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 5, 'Siyam ' ))  db.commit()    result = db.execute('select friend from friends order by friend')  for row in result :      print (row) |

### UPDATE:

আমরা যদি ডেটা বেজে কোন ডেটা ভুল লিখি বা আমাদের ডেটা আপডেট করতে হয়, তাহলে আমরা ডেটা আপডেট করতে পারি। যেমন আমরা যদি উপরের ডেটাতে Shuvo নামটি আপডেট করে Sammo রাখি, তার জন্য আমাদের SQL কমান্ড হবেঃ

|  |  |
| --- | --- |
| 1 | update friends set friend = ? where i = ?', ( 'Sammo ', 1) |

where দিয়ে কোন ইনডেক্স এর ডেটা আপডেট করব, তা ঠিক করে দিয়েছি। সম্পুর্ণ প্রোগ্রামঃ

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | import sqlite3    db = sqlite3.connect('friends.db')    db.execute('drop table if exists friends')    db.execute('create table friends (i int, friend text)')    db.execute('insert into friends(i, friend) values (?,?)', ( 1, 'Shubo ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 2, 'Tuhin ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 3, 'Tanvir ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 4, 'Dipu ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 5, 'Siyam ' ))  db.commit()    print ("nBefore Update: n" )  result = db.execute('select \* from friends')  for row in result :      print (row)    db.execute('update friends set friend = ? where i = ?', ( 'Sammo ', 1) )  db.commit()    print ("n After Update: n" )  result = db.execute('select \* from friends')  for row in result :      print (row) |

### DELETE:

ডিলেট করার সময় বলে দিতে হবে আমরা কি ডিলেট করতে চাই। যেমন একটি row ডিলেট করতে চাইলে লিখব এভাবেঃ

|  |  |
| --- | --- |
| 1 | delete from friends where i = 1 |

সম্পুর্ণ প্রোগ্রামঃ

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | import sqlite3    db = sqlite3.connect('friends.db')    db.execute('drop table if exists friends')    db.execute('create table friends (i int, friend text)')    db.execute('insert into friends(i, friend) values (?,?)', ( 1, 'Shubo ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 2, 'Tuhin ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 3, 'Tanvir ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 4, 'Dipu ' ))  db.execute('insert into friends(i, friend) values (?,?)', ( 5, 'Siyam ' ))  db.commit()    print ("nBefore delete: n" )  result = db.execute('select \* from friends')  for row in result :      print (row)    db.execute('delete from friends where i = 1' )  db.commit()    print ("n After delete: n" )  result = db.execute('select \* from friends')  for row in result :      print (row) |

ডেটাবেজ এর প্রধান চারটি অপারেশন হচ্ছে Create, Read, Updated, Delete. সংক্ষেপে CRUD. এ চারটি অপারেশন হচ্ছে মৈলিক। আরো অনেক কিছুই করা যায়। SQL অপারেশন বলে। SQL এর উপর বিস্তারিত পড়ে সে গুলো জানা যাবে। w3school থেকে SQL টিউটোরিয়াল বা ইউটিউব থেকে SQL + Python এর ভিডিও টিউটোরিয়াল দেখে বিস্তারিত শিখে নেওয়া যাবে।

## ডাটা স্ট্রাকচার ও অ্যালগরিদম এবং পাইথন

[**সহজ ভাষায় পাইথন ৩**](http://bit.ly/2mk5YOB) বইটি পড়ে আমরা যারা পাইথনের অনেকটাই শিখে ফেলেছি (অন্ততপক্ষে বেসিকটুকু শিখে ফেলেছি), আমাদের এখন নতুন করে ভাবতে বসতে হবে। আমরা কতটুকু জ্ঞান সত্যিকার অর্থেই অর্জন করতে পারলাম? এটা বোঝার একটা সহজ উপায় আছে। উপায়টা হল, বিভিন্ন গাণিতিক সমস্যাকে প্রোগ্রামিংয়ের মাধ্যমে সমাধান করার চেষ্টা করা। কিন্তু কথা হল এত সমস্যা পাব কোথায় আর আমরা যে সমাধানটা বের করব সেটা আসলেই সঠিক সমাধান কিনা সেটা বুঝব কিভাবে? ভয় নাই! এইজন্য রয়েছে অনলাইন জাজ (Online Judge)।

অনলাইন জাজ হল এমন একটা অনলাইন সিস্টেম যেখানে কোন প্রোগ্রামকে কোন সমস্যার বিপরীতে অটোমেটিকালি টেস্ট করে দেখা হয় যে সমাধানরা শতভাগ সঠিক আছে কিনা। বিভিন্ন অনলাইন জাজে প্রচুর প্রোগ্রামিং সমস্যা (সাধারণ প্রোগ্রামিং প্রব্লেম বলা হয়) পাওয়া যায়। এসব সমস্যা সমাধান করে উত্তরটা নির্দিষ্ট জায়গায় সাবমিট করলেই জাজ আমাদের জানিয়ে দেবে আমাদের সমাধানটা সঠিক হয়েছে কিনা। পাইথন সাপোর্ট করে এমন কতগুলো জনপ্রিয় অনলাইন জাজ হল:

* [HackerRank](https://www.hackerrank.com/)
* [HackerEarth](https://www.hackerearth.com/practice/)
* [URI](https://www.urionlinejudge.com.br/)
* [UVA](https://uva.onlinejudge.org/)
* [SPOJ](http://www.spoj.com/info/)
* [Codeforces](http://codeforces.com/)
* [TopCoder](https://www.topcoder.com/)

প্রথম ও শেষের দুইটা মূলত কন্টেস্ট প্লাটফর্ম, মানে এসব জায়গায় প্রোগ্রামিং নিয়ে বিভিন্ন অনলাইন প্রতিযোগীতার আয়োজন করা হয় যাতে সবাই অংশ নিতে পারে। তবে কন্টেস্ট প্লাটফর্ম হলেও এসব জায়গায় সলভ করার মত বহুত প্রব্লেম (প্রাকটিস প্রব্লেম নামে অভিহিত) পাওয়া যায়। কিন্তু আমাদের প্রশ্ন হল, এত এত অনলাইন জাজের ভিড়ে আমরা শুরু করব কোনটা দিয়ে?

শুরুর দিকে আমরা HackerRank, HackerEarth ও URI ব্যবহার করতে পারি। HackerEarth ও URI তে বিগিনার, ডাটা স্ট্রাকচার, অ্যালগরিদম এইভাবে প্রব্লেম বিভিন্ন সেকশনে ভাগ করা আছে। অবশ্য পাইথনের ক্ষেত্রে শুরুটা HackerRank দিয়ে করলে ভাল হয়। কারণ, এর প্রব্লেম সেকশনে পাইথন বিভাগে পাইথনের প্রতিটা টপিকের জন্য আলাদা আলাদা প্রব্লেম আছে। প্রোগ্রামিংয়ের বেসিক শেখা শেষ হয়ে গেলে আমরা সবগুলো টপিকের প্রব্লেম সলভ করার চেষ্টা করতে পারি। যে টপিকের প্রব্লেম সলভ করতে বেশি হিমশিম খাব, বুঝে নেব ঐ টপিকে আমরা দুর্বল। সেজন্য বই/অফিসিয়াল ডকুমেন্টেশন খুলে আবার ঐ টপিকের আগা-গোড়া ভালমত বুঝে বুঝে পড়ব। বুঝতে না পারলে বড় কারো সহায়তা নেব। আজকাল ফেসবুকে প্রোগ্রামিংয়ের বিভিন্ন গ্রুপ রয়েছে যেখানে বাংলা ভাষায় নিজের সমস্যার দ্রুত সমাধান পাওয়া যায়। পাইথন নিয়ে বাংলাদেশের সবচেয়ে বড় প্রোগ্রামিং গ্রুপ হল [পাইথন বাংলাদেশ](https://web.facebook.com/groups/pythonbd/)। এছাড়া রয়েছে [এসো প্রোগ্রামিং শিখি](https://web.facebook.com/groups/abcProgramming/) ও [প্রোগ্রামিং প্রব্লেম](https://web.facebook.com/groups/bengaliprogramming/)। এসব গ্রুপে অত্যন্ত চমৎকারভাবে বাংলা/ইংরেজি ভাষায় নিজেদের সমস্যার কথা সংক্ষেপে কিন্তু পরিপূর্ণভাবে তুলে ধরব আমরা। কেউ না কেউ অবশ্যই আমাদের সমস্যার সমাধান করে দেবেন। তখন তাকে প্রাণঢালা ধন্যবাদ জানিয়ে আবার প্রব্লেম সলভিংয়ে লেগে যাব আমরা।

যাহোক, HackerRank এর প্রব্লেম সেকশনের পাইথন বিভাগের সব প্রব্লেম সলভ করা হয়ে গেলে আমরা URI তে চলে যাব। URI এর প্রব্লেম সেকশানের বিগিনার ক্যাটাগরিতে ১৮৮ টা প্রোগ্রামিং প্রব্লেম রয়েছে। সবগুলো একের পর এক শেষ করে ফেলব। আর তারপর লাফ দেব ডাটা স্ট্রাকচারের সমুদ্রে।

### ডাটা স্ট্রাকচার

ডাটা স্ট্রাকচার হল ডাটা অর্গানাইজ ও স্টোর করার স্পেশালাইজড ফরম্যাট যাতে পরবর্তীতে এসব ডাটা সর্বোত্তমভাবে ফিরে পাওয়া যায় ও বিভিন্ন কাজে ব্যবহার করা যায়। কিন্তু আমাদের জন্য ডাটা স্ট্রাকচার কতটা গুরুত্বপূর্ণ?

কম্পিউটারে ডাটা নিয়ে আমরা সাধারণত তিন ধরনের কাজ করি - (১) ডাটা ইনপুট নিই, (২) ডাটা প্রসেস করি ও (৩) ডাটা আউটপুট দিই। আর ডাটা স্ট্রাকচারের উদ্দেশ্য হল এই তিন ধরনের কাজকেই অপটিমাইজ করা বা করতে সাহায্য করা। ধরা যাক, আমরা আমাদের ব্যক্তিগত লাইব্রেরিতে ১০ হাজার বই রাখলাম। বইগুলো যদি আমরা নির্দিষ্ট প্যাটার্ন বা কোন পদ্ধতি অনুসরণ না করে সেলফে রাখি তাহলে সময়কালে একটা বই খুঁজে বের করতে আমাদের কিন্তু ইহজনম পার হয়ে যাবে। এইসব ঝামেলা থেকে মুক্তি পাবার জন্য ডাটা স্ট্রাকচার জানা দরকার আমাদের।

লিস্ট, টাপল, সেট, ডিকশনারি ছিল পাইথনের বিল্ট-ইন ডাটা স্ট্রাকচার। তবে এসবের পাশাপাশি আমাদেরকে আরো কিছু ডাটা স্ট্রাকচার সম্পর্কে জানতে হবে। কমন বা না জানলেই নয় এরকম কিছু ডাটা স্ট্রাকচার হল:

* Stack
* Queue
* Linked List
* Binary Search Tree
* Heaps
* AVL Tree
* B Tree
* B+ Tree
* Binary Indexed Tree
* Trie
* Disjoint-set
* Segment Tree

পাইথন ব্যবহার করে ডাটা স্ট্রাকচার শেখার জন্য আমরা এই [অনলাইন কোর্সটা](https://www.coursera.org/learn/python-data) করতে পারি। ইনিভার্সিটি অব মিশিগানের স্কুল অব ইনফরমেশনের সহযোগী অধ্যাপক Charles Severance এই কোর্সটা পড়িয়েছেন। শিক্ষক হিসেবে তিনি অসাধারণ। সবচেয়ে বড় কথা হল, পড়ানোর ক্ষেত্রে নিজস্ব ভঙ্গিমা রয়েছে তাঁর। কোর্সের ভাষা ইংরেজি। আর কোর্সটা লেকচার ভিডিও হিসেবে দেখতে হবে।

বইয়ের মধ্যে সবচেয়ে ভাল হল [Problem Solving with Algorithms and Data Structures using Python](https://interactivepython.org/runestone/static/pythonds/index.html)। এই বইটা সম্পূ্র্ণ যে পড়বে তার ডাটা স্ট্রাকচার ও অ্যালগরিদমের যাবতীয় বেসিক ক্লিয়ার হয়ে যাবে। বইটাতে যথাযথ উদাহরণও যোগ করা হয়েছে। বইটির ভাষা ইংরেজি।

ডাটস স্ট্রাকচার ও অ্যালগরিদম ভিজুয়ালাইজ (দৃষ্টিগোচর) করার একটা অসাধারণ সাইট হল [এটা](http://www.cs.usfca.edu/~galles/visualization/Algorithms.html)। আমরা যেসব ডাটা স্ট্রাকচার থিওরেটিকালি শিখব, সেগুলো এই সাইটে ভিজুয়ালাইজ করে দেখানো হয়েছে।

বাংলায় ডাটা স্ট্রাকচার ও অ্যালগরিদম শেখার একটা চমৎকার জায়গা হল [শাফায়েত আশরাফ ভাইয়ের ব্লগ](http://www.shafaetsplanet.com/planetcoding/?page_id=2804)। যদিও ব্লগে প্রতিটা উদাহরণ সুডো কোডেই বেশি দেখানো হয়েছে তবে একটা টপিক শেখার পর সেটার সুডো কোডকে ফলো করে পাইথনে ইমপ্লিমেন্ট করার চেষ্টা করা যেতেই পারে। (Pseudocode হল একটা অ্যালগরিদম বা প্রোগ্রাম কিভাবে ধাপ-বাই-ধাপ কাজ করে তার সংক্ষিপ্ত বর্ণনা।)

মোটামুটি শেখা হয়ে গেলে HackerRank এর প্রব্লেম সেকশনের ডাটা স্ট্রাকচার বিভাগের প্রতিটা প্রব্লেম সলভ করার চেষ্টা করব আমরা। URI অনলাইন জাজের প্রব্লেম সেকশনে ডাটা স্ট্রাকচারের উপর ১১০ টা প্রব্লেম রয়েছে। এগুলোও সলভ করার চেষ্টা করব।

### অ্যালগরিদম

ইতিমধ্যেই আমরা অ্যালগরিদম শব্দটার সঙ্গে পরিচিত হয়ে গিয়েছি। অ্যালগরিদম হল কতগুলো অপারেশনের সেট। আরো ভালভাবে বলতে গেলে, কোন একটা সমস্যাকে সবচেয়ে উৎকৃষ্ট যে পদ্ধতিতে সমাধান করা যায় তাকেই অ্যালগরিদম বলে। এগুলো বল গণিত দ্বারা প্রমাণিত পদ্ধতি। কোন একটা সমস্যা সাধারণ বুদ্ধি প্রয়োগ করে সমাধান করলে সেটা যতটা ইফেক্টিভ হয়, অ্যালগরিদম ব্যবহার করে সলভ করলে ঢের বেশি ইফেক্টিভ হয়।

সুতরাং মদ্দা কথা হল আমাদেরকে অ্যালগরিদম শিখতে হবে। দুনিয়ায় বহুত অ্যালগরিদম আছে। তবে কমন বা না জানলেই নয়, এরকম কিছু অ্যালগরিদম হল:

* Bubble Sort
* Insertion Sort
* Selection Sort
* Quick Sort
* Heap Sort
* Depth First Search (DFS)
* Breadth First Search (BFS)
* A\* Search
* Hill Climbing

পূ্র্বে উল্লেখিত Problem Solving with Algorithms and Data Structures using Python বইটা দিয়েই শেখা শুরু করতে পারি আমরা। বাংলায় শিখতে চাইলে শাফায়েত ভাইয়ের ‘গ্রাফ অ্যালগরিদম’ বইটা কিনে নেয়া যেতে পারে। তবে সেক্ষেত্রে প্রতিটা অ্যালগরিদম নিজে নিজে পাইথনে ইমপ্লিমেন্ট করার চেষ্টা করতে হবে।

তবে যেখান থেকেই শিখি না কেন প্রব্লেম সলভিংয়ে জোড় দিতে হবে আমাদের। বিভিন্ন কন্টেস্টে অংশ নিতে হবে। বিশেষত আমাদের ভিতর যাদের গুগল বা মাইক্রোসফটের মত বড়সড় কোম্পানিতে চাকরি করার প্রবল ইচ্ছা আছে। এসব কোম্পানি প্রব্লেম সলভিংয়ে সিদ্ধহস্তদের অনেক পছন্দ করে। তাছাড়া এসব কোম্পানির ইন্টারভিউতেও বিভিন্ন প্রব্লেম সলভ করতে দেয়া হয়।

ডাটা স্ট্রাকচার ও অ্যালগরিদম শেখার একটা ব্যক্তিগত তালিকা আছে আমার। আপনারাও দেখতে পারেন: <https://blog.maateen.me/bn/data-structures-and-algorithms/>।

মোটামুটি এই ছিল কথাবার্তা। তো, এখন কাজ কি? প্রব্লেম সলভিং চালিয়ে যাওয়া।

### ডাটা স্ট্রাকচার : স্ট্যাক

মার্চ 26, 2017 তারিখে, বিভাগ: [প্রোগ্রামিং](https://blog.maateen.me/bn/categories/%e0%a6%aa%e0%a7%8d%e0%a6%b0%e0%a7%8b%e0%a6%97%e0%a7%8d%e0%a6%b0%e0%a6%be%e0%a6%ae%e0%a6%bf%e0%a6%82), [পাইথন](https://blog.maateen.me/bn/categories/%e0%a6%aa%e0%a6%be%e0%a6%87%e0%a6%a5%e0%a6%a8), [ডাটা স্ট্রাকচার](https://blog.maateen.me/bn/categories/%e0%a6%a1%e0%a6%be%e0%a6%9f%e0%a6%be-%e0%a6%b8%e0%a7%8d%e0%a6%9f%e0%a7%8d%e0%a6%b0%e0%a6%be%e0%a6%95%e0%a6%9a%e0%a6%be%e0%a6%b0)

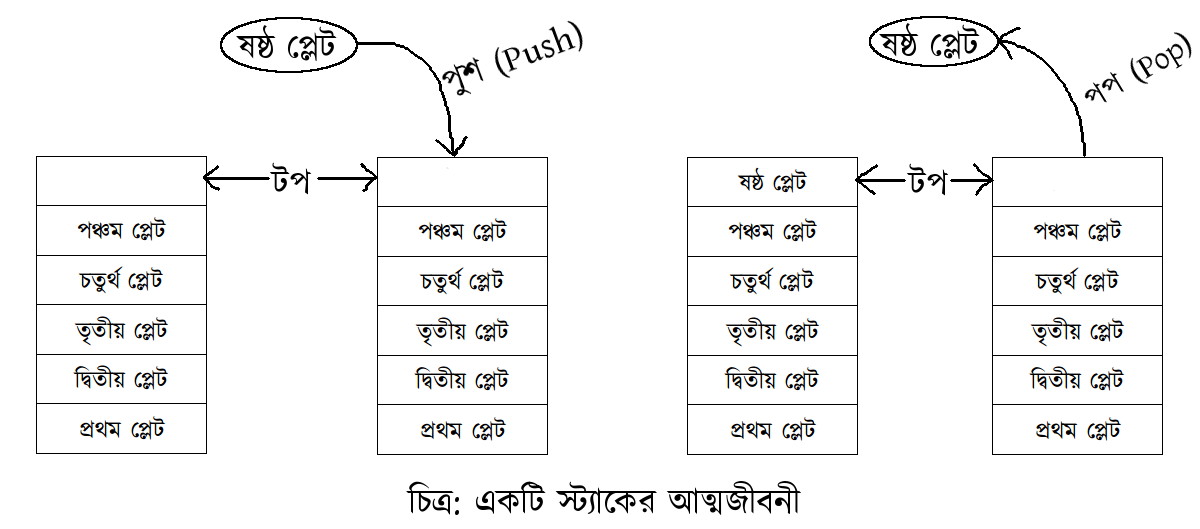
ডাটা স্ট্রাকচার নিয়ে কথা বলতে গেলে স্ট্যাক নিয়ে কথা একেবারে না বললেই নয়। শব্দটা আমি প্রথম শুনেছিলাম আমার এক বন্ধুর কাছ থেকে। ঘটনাটা বেশ মজার। কোন এক বোরিং ক্লাসে বসে ছিলাম। ভাবলাম সামনের বন্ধুটাকে কলম দিয়ে খুঁচিয়ে কিছুটা বিনোদন পাওয়া যেতে পারে। তো যেই ভাবা সেই কাজ।

সাধের ম্যাটাডোর অরবিট কলমটাকে সামনের দিকে বাগিয়ে বন্ধুর পশ্চাৎদেশে জোরসে একটা গুঁতো মারলাম। কিন্তু সে আমার গুঁতোতে তেমন একটা রেসপন্স করল না। আমি ভাবলাম রেসপন্স না করার দুটো কারণ হতে পারে। এক, ও ক্লাসের সবচেয়ে মনোযোগী ছাত্র আর দুই, ওর শরীরে গন্ডারের জিন রয়েছে। পরীক্ষা-নিরীক্ষা চালিয়ে আরো নিশ্চিত হবার জন্য আগের চেয়ে খানিক জোরসে আবার একটা গুঁতো মারলাম। এবার ব্যাটা গন্ডারের বংশধর ভয়ানক রেসপন্স করল। পিছন ফিরে চোখ রাঙিয়ে আমাকে হুমকি দিল –

আর একবার যদি গুঁতা মারস; তাইলে তোরে মাইরা, তোর হাড্ডি ভাইঙ্গা স্ট্যাক কইরা রাখমু।

এরকম হুমকির পরে গুঁতো কর্মসূচীতে আর তেমন একটা উৎসাহ পেলাম না। সেদিনের মত ইস্তফা দিয়ে বাসায় চলে আসলাম। বাসায় এসে ডিকশনারি ঘেঁটে দেখলাম, স্ট্যাক মানে স্তুপ। যাহোক, অনেক গপ্পোসপ্পো হল। এবার স্ট্যাক নিয়ে আসল কথায় আসা যাক।

সংক্ষেপে, স্ট্যাক হল স্তুপ। আর বিশদভাবে বলতে গেলে, স্ট্যাক হল কতগুলো আইটেমের এমন এক কাঠামোবদ্ধ (স্ট্রাকচারড - structured) সংগ্রহশালা (কালেকশন - collection) যেখানে নতুন আইটেমের সংযোজন (পুশ - push) বা পুরনো আইটেমের অপসারণ (পপ - pop) সংগ্রহশালার একই প্রান্তে হয়। ব্যাপারটা ঠিক বোঝা গেল না, তাই না? কোন ব্যাপার না। একটা উদাহরণ দিলেই ব্যাপারটা পরিষ্কার হয়ে যাবে।



পাঁচটা খাবার প্লেটের একটা স্তুপ (স্ট্যাক) কল্পনা করা যাক। আমরা যদি এই স্তুপে নতুন একটা প্লেট (আইটেম) সংযোজন (পুশ) করতে চাই তবে ষষ্ঠতম প্লেটটা স্তুপের একেবারে উপরে রাখতে হবে। এখন আমাদের কাছে ছয়টা প্লেটের একটা স্তুপ আছে। এবার যদি এই স্তুপ থেকে একটা প্লেট (আইটেম) অপসারণ (পপ) করতে চাই তবে ষষ্ঠতম প্লেটটাই কিন্তু অপসারণ করতে হবে। খেয়াল করলে দেখব, যে একই প্রান্তে এই কাজগুলো করছি সেটার একটা নাম দেয়া যায় – উপর বা টপ (Top)। আর এর অপর প্রান্তকে বলা যায় নিচ বা বেইস (Base)।

আমাদের ভিতর যারা অত্যন্ত বুদ্ধিমান তারা হয়ত এতক্ষণে বুঝে ফেলেছি যে এটা একটা লাস্ট-ইন-ফার্স্ট-আউট (LIFO) ডাটা স্ট্রাকচার। কারণ, আমরা সবার শেষে যে আইটেমটা সংযোজন (পুশ) করছি সেটাই কিন্তু সবার আগে অপসারণ (পপ) করছি। মানে, ঢুকছে সবার শেষে কিন্তু বের হচ্ছে সবার আগে। কি মজা!

#### অপারেশন

এক নজরে এখন আমরা স্ট্যাকের সকল অপারেশন দেখব। সাধারণ পাঁচ ধরনের ফাংশন বা মেথডের মাধ্যমে এই অপারেশনগুলি সম্পাদিত হয়।

#### push(item)

স্ট্যাকের উপরে (টপে) নতুন আইটেম সংযোজন করার জন্য এই ফাংশন বা মেথডটি ব্যবহার করা হয়। আর্গুমেন্ট বা প্যারামিটার হিসেবে আইটেমটাকে গ্রহণ করলেও এটি কোন কিছু রিটার্ন করে না।

#### pop()

এই ফাংশন বা মেথডটি আর্গুমেন্ট বা প্যারামিটার হিসেবে কোন কিছু গ্রহণ না করলেও স্ট্যাকের একেবারে উপরের আইটেমটাকে রিটার্ন করে। এর পাশাপাশি উক্ত আইটেমটাকে স্ট্যাক থেকে অপসারণও (রিমুভ) করে।

#### peek()

এই ফাংশন বা মেথডটি অনেকটা pop()-এর মতই। শুধু পার্থক্য হল এটি pop() এর মত স্ট্যাক থেকে আইটেম অপসারণ (রিমুভ) করে না।

#### is\_empty()

এটি একটি বুলিয়ান ফাংশন (বা মেথড)। স্ট্যাক খালি কিনা সেটি চেক করে True বা False রিটার্ন করে। এরও কোন প্যারামিটার নেই।

#### size()

এই ফাংশন (বা মেথড) কোন আর্গুমেন্ট বা প্যারামিটার গ্রহণ করে না এবং স্ট্যাকের মোট আইটেম সংখ্যা রিটার্ন করে।

অপারেশনগুলো সবাই বুঝতে পারলাম? দুই-একজন হয়ত পুরোপুরিভাবে বুঝতে পারিনি। এতে অবশ্য দুশ্চিন্তা করার মত কিছু নেই। বেশ কয়েকবার পড়লে এবং স্ট্যাক ইমপ্লিমেন্টেশন করার সময় বেসিক আরো ক্লিয়ার হয়ে যাবে।

#### ইমপ্লিমেন্টেশন

যেকোন স্ট্রাকচারড বা অবজেক্ট-অরিয়েন্টেড প্রোগ্রামিং ল্যাঙ্গুয়েজেই আমরা স্ট্যাককে ইমপ্লিমেন্ট করতে পারি। শুধু খেয়াল রাখতে হবে, প্রোগ্রামটিতে যেন স্ট্যাকের সকল অপারেশন করা যায়। স্ট্যাকের আইটেমগুলো স্টোর করার জন্য আমাদের একটি অ্যারে বা অ্যারের মত কিছু লাগবে। অ্যারের মত এইজন্য বললাম কারণ পাইথনে অ্যারে নেই কিন্তু অ্যারের মত (অ্যারের চেয়েও অ্যাডভান্সড) লিস্ট রয়েছে। যাহোক, অ্যারে বা লিস্টের ইনডেক্স জিরো হবে স্ট্যাকের বেইস আর সর্বোচ্চ ইনডেক্স হবে স্ট্যাকের টপ। যখন বেইস আর টপ সমান হবে তখন বুঝতে হবে স্ট্যাকের ভিতরে কিছু নেই, এটি এখন খালি (empty)। এরপর উপরে বলা পাঁচ ধরনের ফাংশন (বা মেথড) ইমপ্লিমেন্ট করতে হবে।

এবার আমরা স্ট্যাককে পাইথনে (পাইথন ৩.x) ইমপ্লিমেন্ট করব। সবাই সবারমত করে চেষ্টা করব। উদাহরণ হিসেবে [এটা দেখতে পারি](https://github.com/maateen/data-structure-practice/blob/master/Stack.py)।

প্রোগ্রামটা আর ব্যাখ্যা করলাম না। আশা করি, সবাই বুঝতে পেরেছি। আচ্ছা, বুঝলাম কিনা সেটা কিভাবে টেস্ট করে দেখা যায়? এক মগ কফি খেয়ে সবাই এখন [হ্যাকারর‍্যাংক](https://www.hackerrank.com/domains/data-structures/stacks/page:1) ও [হ্যাকারআর্থের](https://www.hackerearth.com/practice/data-structures/stacks/basics-of-stacks/practice-problems/) স্ট্যাক রিলেটেড প্রব্লেমগুলো সলভ করব। আর সলভ করার আগ পর্যন্ত কারো সাথে কোন কথা নাই।

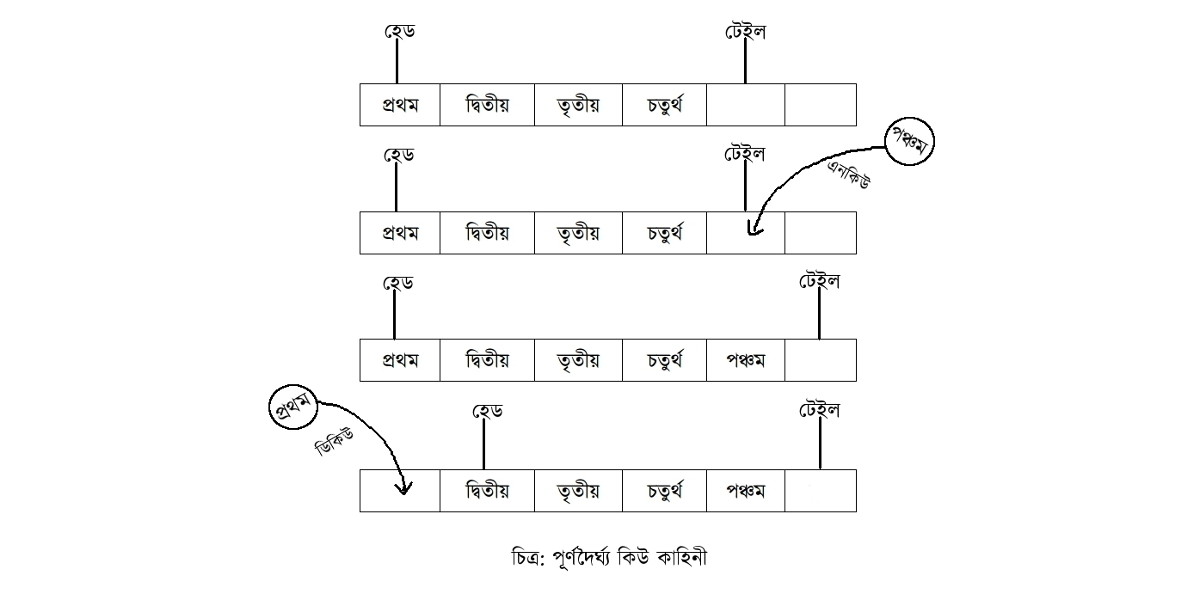
### ডাটা স্ট্রাকচার : কিউ

মার্চ 28, 2017 তারিখে, বিভাগ: [প্রোগ্রামিং](https://blog.maateen.me/bn/categories/%e0%a6%aa%e0%a7%8d%e0%a6%b0%e0%a7%8b%e0%a6%97%e0%a7%8d%e0%a6%b0%e0%a6%be%e0%a6%ae%e0%a6%bf%e0%a6%82), [পাইথন](https://blog.maateen.me/bn/categories/%e0%a6%aa%e0%a6%be%e0%a6%87%e0%a6%a5%e0%a6%a8), [ডাটা স্ট্রাকচার](https://blog.maateen.me/bn/categories/%e0%a6%a1%e0%a6%be%e0%a6%9f%e0%a6%be-%e0%a6%b8%e0%a7%8d%e0%a6%9f%e0%a7%8d%e0%a6%b0%e0%a6%be%e0%a6%95%e0%a6%9a%e0%a6%be%e0%a6%b0)

[স্ট্যাকের](https://bn.maateen.me/data-structure-stack/) (Stack) মতই আরেকটি লিনিয়ার (Linear) ডাটা স্ট্রাকচার হল কিউ (Queue)। লিনিয়ার ডাটা স্ট্রাকচার বলতে বুঝায় যেখানে আইটেমগুলো ধারাবাহিকভাবে রয়েছে, যেমন: স্ট্যাক, কিউ, লিংকড (Linked) লিস্ট। বাংলায় কিউকে আমরা সারি বলতে পারি। তবে বুঝানোর সুবিধার্থে আমরা কিউ বলেই আপাতত চালিয়ে নেব।

কিউ হল কতগুলো আইটেমের এমন এক ধারাবাহিক সংগ্রহশালা (কালেকশন - collection) যেখানে নতুন আইটেমের সংযোজন (এনকিউ – enqueue) সংগ্রহশালার এক প্রান্তে আর পুরনো আইটেমের অপসারণ (ডিকিউ - dequeue) ঠিক তার বিপরীত প্রান্তে হয়। বোঝার সুবিধার্থে, যে প্রান্তে নতুন আইটেমের সংযোজন হয় সে প্রান্তকে আমরা পিছনের অংশ বা রিয়ার (rear) অথবা টেইল (tail - লেজ) বলতে পারি। আর যে প্রান্তে পুরনো আইটেমের অপসারণ হয় সে প্রান্তকে আমরা সামনের অংশ বা ফ্রন্ট (front) অথবা হেড (head - মাথা) বলতে পারি। বেশ গোলমেলে ব্যাপার-স্যাপার, তাই না? দুশ্চিন্তা করার কোন কারণ নেই। একটা গল্প বললেই বিষয়টা পরিষ্কার হয়ে যাবে।

বিদ্যুৎ বিল দেবার শেষ তারিখ চলে এসেছে, অথচ তখনো আমাদের বিদ্যুৎ বিল দেয়া হয়নি। অন্যদিকে কায়িক পরিশ্রম না করার কারণে আমার শরীরও ফুলে ঢোল হয়ে গিয়েছে। আম্মাজান দুয়ে দুয়ে চার মিলিয়ে ফেললেন। হাতে বিদ্যুৎ বিল ধরিয়ে দিয়ে আমাকে ব্যাংকে পাঠিয়ে দিলেন। যাতায়াত খরচ না দেয়ার কারণে হেঁটে হেঁটেই ব্যাংকে আসতে হল। গিয়ে দেখলাম, বিদ্যুৎ বিলের বুথের সামনে চারজনের একটা লাইন। তড়িঘড়ি করে লাইনে দাঁড়িয়ে পাঁচ নাম্বারে আমার অবস্থান নিশ্চিত করলাম। বুথের ভদ্রলোক একজন-একজন করে বিল নিচ্ছেন। সময় কাটানোর জন্য কি করা যেতে পারে ভাবতে লাগলাম। শেষমেষ গুনগুন করে গান গাওয়া শুরু করলাম,“কবে আইবে আমার পালা রে…।” আশ্চর্য ফলাফল পেলাম। পুরো গান শেষ হতে না হতেই আমার পালা চলে এলো। ততক্ষণে অবশ্য আমার পিছনে আরো দুইজন এসে গিয়েছে। কিন্তু আমার আগে বিল দেয়ার সুযোগ নেই তাদের। ব্যাংকের নিয়ম অত্যন্ত কড়া। আমার পরেই তারা সুযোগ পাবে। অবশেষে বিল দিয়ে নাচতে নাচতে বাসায় চলে আসলাম। বিল দিতে গিয়ে কিউ ডাটা স্ট্রাকচারের বাস্তবিক উদাহরণ পেয়েছি - এটাই নাচা-নাচির কারণ।



উপরের চিত্রে আমরা একটি পূর্ণদৈর্ঘ্য কিউ কাহিনী দেখতে পাচ্ছি। প্রাথমিকভাবে, **প্রথম জন** হচ্ছে আমাদের কিউয়ের হেড আর **চতুর্থ জনের** ডান পাশের ঘরটা হচ্ছে টেইল। **পঞ্চম জন** (আইটেম)-কে আমরা যদি আমাদের কিউয়ে এনকিউ (সংযোজন) করতে চাই তবে তা টেইলে করতে হবে। আর **পঞ্চম জন** (আইটেম) এনকিউ হয়ে যাবার পরে টেইল কিন্তু এক ঘর ডানে সরে আসবে। অর্থাৎ **পঞ্চম জনের** ডান পাশের ঘরটা হবে নতুন টেইল। আবার ডিকিউ (অপসারণ) করার সময় হেডকে সবার আগে ডিকিউ করতে হবে। ঘটনাচক্রে, হেডে রয়েছে **প্রথম জন**, তাই **প্রথম জনকেই** সবার আগে ডিকিউ করতে হবে। **প্রথম জনকে** ডিকিউ করা হয়ে গেলে উক্ত ঘর ফাঁকা হয়ে যাবে, তাই হেড এক ঘর ডানে সরে আসবে। তখন নতুন হেডে থাকবে **দ্বিতীয় জন**। আচ্ছা, এই জন কি অভিনেতা জন? নাকি এই জন দিয়ে মানুষজন বুঝানো হয়েছে? এটা কুইজ। যে সবার আগে উত্তর দিতে পারবে সে একটা হাওয়াই চকলেট পাবে।

উপরের চিত্রটা দেখে একটা বিষয় তো আমরা সবাই কম-বেশি বুঝে ফেলেছি যে, কিউ একটি ফার্স্ট-ইন-ফার্স্ট-আউট (FIFO) ডাটা স্ট্রাকচার। কারণ, সবার আগে যে ঢুকছে সেই সবার আগে বের হচ্ছে। অনেকটা ‘আগে আসলে আগে পাবেন’ (ফার্স্ট-কাম-ফার্স্ট-সার্ভড) টাইপের অবস্থা। খুব মজার, তাই না?

#### অপারেশন

এক নজরে আমরা এখন কিউয়ের সকল অপারেশন দেখব। সাধারণত চার ধরনের ফাংশন বা মেথডের মাধ্যমে এই অপারেশনগুলো সম্পাদিত হয়।

#### enqueue(item)

কিউয়ের রিয়ার বা টেইলে নতুন কোন আইটেম সংযোজন করার জন্য এই ফাংশন বা মেথডটি ব্যবহার করা হয়। আর্গুমেন্ট বা প্যারামিটার হিসেবে আইটেমটাকে গ্রহণ করলেও এটি কোন কিছু রিটার্ন করে না।

#### dequeue()

এই ফাংশন বা মেথডটি আর্গুমেন্ট বা প্যারামিটার হিসেবে কোন কিছু গ্রহণ না করলেও কিউয়ের হেডে থাকা আইটেমটাকে রিটার্ন করে। এর পাশাপাশি উক্ত আইটেমটাকে কিউ থেকে অপসারণও (রিমুভ) করে।

#### is\_empty()

এটি একটি বুলিয়ান ফাংশন (বা মেথড)। কিউ খালি কিনা সেটি চেক করে True বা False রিটার্ন করে। এরও কোন প্যারামিটার নেই।

#### size()

এই ফাংশন (বা মেথড) কোন আর্গুমেন্ট বা প্যারামিটার গ্রহণ করে না এবং কিউয়ের মোট আইটেম সংখ্যা রিটার্ন করে।

অপারেশনগুলো সবাই বুঝতে পারলাম? দুই-একজন হয়ত পুরোপুরিভাবে বুঝতে পারিনি। এতে অবশ্য দুশ্চিন্তা করার মত কিছু নেই। বেশ কয়েকবার পড়লে এবং কিউ ইমপ্লিমেন্টেশন করার সময় বেসিক আরো ক্লিয়ার হয়ে যাবে।

#### ইমপ্লিমেন্টেশন

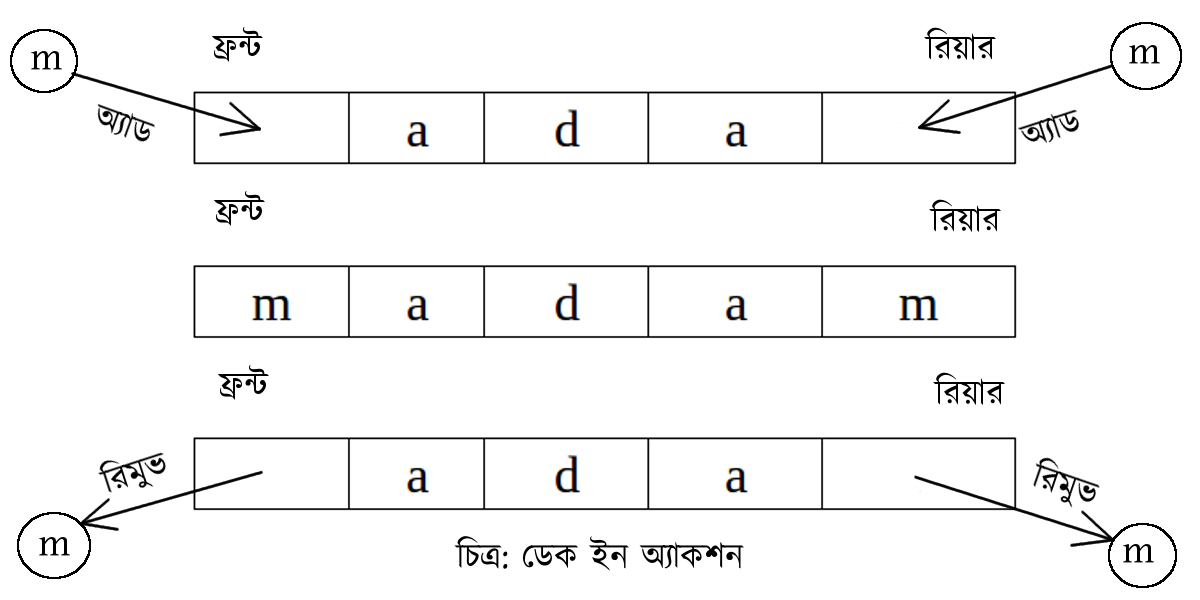
স্ট্যাকের মত কিউ-কেও যেকোন স্ট্রাকচারড বা অবজেক্ট-অরিয়েন্টেড প্রোগ্রামিং ল্যাঙ্গুয়েজেই আমরা ইমপ্লিমেন্ট করতে পারি। শুধু খেয়াল রাখতে হবে, প্রোগ্রামটিতে যেন কিউয়ের সকল অপারেশন করা যায়। কিউয়ের আইটেমগুলো স্টোর করার জন্য আমাদের একটি অ্যারে বা লিস্ট (পাইথনে অ্যারের বদলে লিস্ট রয়েছে) লাগবে। যাহোক, অ্যারে বা লিস্টের ইনডেক্স জিরো হবে কিউয়ের ফ্রন্ট বা হেড আর সর্বোচ্চ ইনডেক্স হবে কিউয়ের রিয়ার বা টেইল। যখন হেড আর টেইলের ইনডেক্স একই হবে তখন বুঝতে হবে কিউয়ের ভিতরে কিছু নেই, এটি এখন খালি (empty)। এরপর উপরে বলা চার ধরনের ফাংশন (বা মেথড) ইমপ্লিমেন্ট করতে হবে।

এবার আমরা কিউ-কে পাইথনে (পাইথন ৩.x) ইমপ্লিমেন্ট করব। সবাই সবার মত করে চেষ্টা করব। উদাহরণ হিসেবে [এটা দেখতে পারি](https://github.com/maateen/data-structure-practice/blob/master/Queue.py)।

### ডাটা স্ট্রাকচার : ডেক

মে 7, 2017 তারিখে, বিভাগ: [প্রোগ্রামিং](https://blog.maateen.me/bn/categories/%e0%a6%aa%e0%a7%8d%e0%a6%b0%e0%a7%8b%e0%a6%97%e0%a7%8d%e0%a6%b0%e0%a6%be%e0%a6%ae%e0%a6%bf%e0%a6%82), [পাইথন](https://blog.maateen.me/bn/categories/%e0%a6%aa%e0%a6%be%e0%a6%87%e0%a6%a5%e0%a6%a8), [ডাটা স্ট্রাকচার](https://blog.maateen.me/bn/categories/%e0%a6%a1%e0%a6%be%e0%a6%9f%e0%a6%be-%e0%a6%b8%e0%a7%8d%e0%a6%9f%e0%a7%8d%e0%a6%b0%e0%a6%be%e0%a6%95%e0%a6%9a%e0%a6%be%e0%a6%b0)

ইতিমধ্যে আমরা [স্ট্যাক](https://bn.maateen.me/data-structure-stack/) (Stack) ও [কিউ](https://bn.maateen.me/data-structure-queue/) (Queue) সম্পর্কে জেনেছি। এগুলো হল লিনিয়ার ডাটা স্ট্রাকচার ও একমুখো সাপ। একমুখো সাপ বললাম কারণ, স্টাক ও কিউতে আইটেম একটিমাত্র প্রান্তে ঢুকতে পারে ও একটিমাত্র প্রান্ত থেকে বের হতে পারে। আরেকটু পরিষ্কার করে বললে, স্ট্যাকের ক্ষেত্রে যে প্রান্তে আইটেম ঢুকবে সেই প্রান্ত দিয়েই বের হবে আর কিউয়ের ক্ষেত্রে ঢুকবে এক প্রান্ত দিয়ে কিন্তু বের হবে অন্য প্রান্ত দিয়ে। লক্ষণীয় বিষয় হল, দুই ক্ষেত্রেই ঢোকার বা বের হবার রাস্তা কেবলমাত্র একটি। কিন্তু ডেক (Deque) হল একটি দুমুখো সাপ। এর দুই মুখেই আইটেম ঢুকতে পারে অথবা দুই মুখ থেকেই আইটেম বের হতে পারে।



মূলত ডাবল-এন্ডেড কিউ (double-ended queue)-এর শর্টফর্ম হল ডেক। ডেক হল কতগুলো আইটেমের এমন এক ধারাবাহিক সংগ্রহশালা (কালেকশন – collection) যেখানে নতুন আইটেমের সংযোজন বা পুরনো আইটেমের অপসারণ সংগ্রহশালার উভয় প্রান্তে হয়। যদি আমরা এক প্রান্তকে সামনের অংশ বা ফ্রন্ট (front) বলি তবে অপর প্রান্তকে পিছনের অংশ বা রিয়ার (rear) বলে অভিহিত করতে পারি। তাহলে আইটেমের সংযোজন বা অপসারণ ফ্রন্ট বা রিয়ার, যেকোন অংশেই হতে পারে। মদ্দা কথা হল, ডেক একটি হাইব্রিড ডাটা স্ট্রাকচার যাতে স্ট্যাক ও কিউ, উভয়ের সুবিধাই বর্তমান।

#### অপারেশন

এক নজরে আমরা এখন ডেকের সকল অপারেশন দেখব। সাধারণত ছয় ধরনের ফাংশন বা মেথডের মাধ্যমে এই অপারেশনগুলো সম্পাদিত হয়।

#### add\_front(item)

ডেকের ফ্রন্টে একটি নতুন আইটেমেট সংযোজন করার জন্য এই ফাংশন বা মেথডটি ব্যবহার করা হয়। প্যারামিটার বা আর্গুমেন্ট হিসেবে আইটেমটাকে গ্রহণ করলেও এটি কোন কিছু রিটার্ন করে না।

#### add\_rear(item)

ডেকের রিয়ারে একটি নতুন আইটেমেট সংযোজন করার জন্য এই ফাংশন বা মেথডটি ব্যবহার করা হয়। প্যারামিটার বা আর্গুমেন্ট হিসেবে আইটেমটাকে গ্রহণ করলেও এটি কোন কিছু রিটার্ন করে না।

#### remove\_front()

এই ফাংশন বা মেথডটি আর্গুমেন্ট বা প্যারামিটার হিসেবে কোন কিছু গ্রহণ না করলেও ডেকের ফ্রন্টে থাকা আইটেমটাকে রিটার্ন করে। এর পাশাপাশি উক্ত আইটেমটাকে ডেক থেকে অপসারণও (রিমুভ) করে।

#### remove\_rear()

এই ফাংশন বা মেথডটি আর্গুমেন্ট বা প্যারামিটার হিসেবে কোন কিছু গ্রহণ না করলেও ডেকের রিয়ারে থাকা আইটেমটাকে রিটার্ন করে। এর পাশাপাশি উক্ত আইটেমটাকে ডেক থেকে অপসারণও (রিমুভ) করে।

#### is\_empty()

এটি একটি বুলিয়ান ফাংশন (বা মেথড)। ডেক খালি কিনা সেটি চেক করে True বা False রিটার্ন করে। এরও কোন প্যারামিটার নেই।

#### size()

এই ফাংশন (বা মেথড) কোন আর্গুমেন্ট বা প্যারামিটার গ্রহণ করে না এবং ডেকের মোট আইটেম সংখ্যা রিটার্ন করে।

একেবারে স্ট্যাক ও কিউয়ের অপারেশনগুলোর মিশেল তাই না? ওহ! সাবধান! এই মিশেল কিন্তু আবার মিশেল ওবামা না।

#### ইমপ্লিমেন্টেশন

স্ট্যাক, কিউয়ের মত ডেকও যেকোন স্ট্রাকচারড বা অবজেক্ট-অরিয়েন্টেড প্রোগ্রামিং ল্যাঙ্গুয়েজেই আমরা ইমপ্লিমেন্ট করতে পারি। শুধু খেয়াল রাখতে হবে, প্রোগ্রামটিতে যেন ডেকের সকল অপারেশন করা যায়। বরাবরের মত ডেকের আইটেমগুলো স্টোর করার জন্য আমাদের একটি অ্যারে বা লিস্ট (পাইথনে অ্যারের বদলে লিস্ট রয়েছে) লাগবে। যাহোক, প্রাথমিকভাবে অ্যারে বা লিস্টের ইনডেক্স জিরোকে ডেকের ফ্রন্ট ধরতে পারি আর ইনডেক্স ওয়ানকে ডেকের রিয়ার ধরতে পারি। যখন ফন্ট আর রিয়ারের ইনডেক্স একই হবে তখন বুঝতে হবে ডেকের ভিতরে কিছু নেই, এটি এখন খালি (empty)। এরপর উপরে বলা ছয় ধরনের ফাংশন (বা মেথড) ইমপ্লিমেন্ট করতে হবে।

এবার আমরা ডেক-কে পাইথনে (পাইথন ৩.x) ইমপ্লিমেন্ট করব। সবাই সবার মত করে চেষ্টা করব। উদাহরণ হিসেবে [এটা দেখতে পারি](https://github.com/maateen/data-structure-practice/blob/master/Deque/Deque.py)।

#### অ্যাপ্লিকেশন

একটা ওয়ার্ড প্যালিনড্রোম (palindrome) কিনা তা চেক করে দেখার জন্য ডেক ব্যবহার করা যায়। কিছু কিছু শব্দ আছে যেগুলোকে আমরা উল্টে-পাল্টে যেভাবেই পড়িনা কেন, শব্দটা বদলায় না। যেমন: civic, level, madam। এগুলোকে প্যালিন্ড্রোম বলে। যাহক, যে শব্দটা চেক করব সেটা একটা ডেকের ভিতর রেখে ডেকের ফ্রন্ট ও রিয়ার থেকে একটা একটা করে স্ট্রিং নিয়ে চেক করে দেখা যেতে পারে ঐ দুটি স্ট্রিং একই কিনা। এভাবে পর্যায়ক্রমে সবগুলো চেক করে দেখা যেতে পারে। তারপরে সিদ্ধান্ত নেয়া যায় - শব্দটি প্যালিন্ড্রোম? নাকি নয়?

from Deque import Deque

# Deque is already defined in Deque.py

def is\_pallindrome(word):

d = Deque()

if len(word) > 1:

for char in word:

d.add\_rear(char)

equal = True

while d.size() > 1 and equal:

first\_char = d.remove\_front()

last\_char = d.remove\_rear()

if first\_char == last\_char:

equal = True

else:

return False

return True

else:

return 'The word should consist of at least two chars.'

if \_\_name\_\_ == '\_\_main\_\_':

while True:

print('Please, input the word to check:')

print(is\_pallindrome(input()))

[পাইথনে](https://www.rokomari.com/book/127980/) লেখা এই প্রোগ্রামটা আর ব্যাখ্যা করলাম না। আশা করি, সবাই বুঝতে পেরেছি। সবশেষে একটা কুইজ - বাংলাদেশের সবচেয়ে জনপ্রিয় দুমুখো সাপের নাম কি?

### ডাটা স্ট্রাকচার : সিঙ্গলি লিংকড লিস্ট

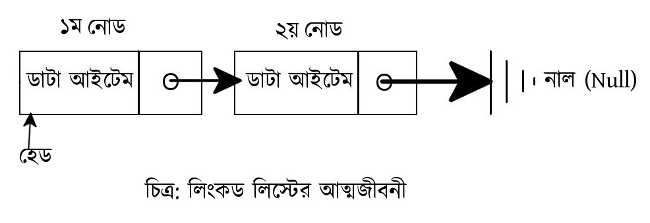
জুলাই 1, 2017 তারিখে, বিভাগ: [ডাটা স্ট্রাকচার](https://blog.maateen.me/bn/categories/%e0%a6%a1%e0%a6%be%e0%a6%9f%e0%a6%be-%e0%a6%b8%e0%a7%8d%e0%a6%9f%e0%a7%8d%e0%a6%b0%e0%a6%be%e0%a6%95%e0%a6%9a%e0%a6%be%e0%a6%b0)

ইতিপূর্বের উদাহরণগুলোতে আমরা পাইথনের বিল্ট-ইন ডাটা স্ট্রাকচার - লিস্টের ব্যবহার দেখে এসেছি। লিংকড লিস্টকে আমরা সেই লিস্টের বিশেষায়িত রূপ বলতে পারি। বিশেষায়িত বলার কারণ হচ্ছে, এই লিস্টের ভিতর কিছু বিশেষ ব্যাপার-স্যাপার রয়েছে।

মোটামুটি চার টাইপের লিংকড লিস্ট রয়েছে: সিম্পল (Simple) বা সিঙ্গলি (Singly) লিংকড লিস্ট, ডাবলি (Doubly) লিংকড লিস্ট, মাল্টিপ্লাই (Multiply) লিংকড লিস্ট ও সার্কুলার (Circular) লিংকড লিস্ট। আজকে আমরা সিঙ্গলি লিংকড লিস্ট সম্পর্কে জানার চেষ্টা করব।

সহজ ভাষায়, সিঙ্গলি লিংকড লিস্ট হল কতগুলো নোডের চেইন বা সমাহার। আরেকটু পুস্তকী ভাষায় বলতে গেলে, সিঙ্গলি লিংকড লিস্ট হল কতগুলো ডাটা এলিমেন্ট বা নোডের ধারাবাহিক সংগ্রহশালা (কালেকশন – collection)। লিনিয়ার সিকুয়েন্স আর কি! এখানে, নোড আসলে একটা বেসিক ইউনিট। প্রতিটি নোডে দুটি ফিল্ড থাকে। প্রথম ফিল্ডে থাকে ডাটা আইটেম আর শেষের ফিল্ডে থাকে পয়েন্টার বা পরবর্তী নোডের লিংক।

ব্যাপারটা সহজে বোঝার জন্য আমরা শিশুবেলায় ফিরে যেতে পারি। সেই স্কুল পালানো দুরন্তপনার দিনগুলো কি মনে আছে আমাদের? বিশেষ করে, হাত ধরাধরি করে দশ-পনেরজন পাশাপাশি দাঁড়িয়ে থাকার দিনগুলো? এটা মনে থাকলেই আপাতত চলবে। যখন আমরা দশ-পনেরজন হাত ধরাধরি করে পাশাপাশি দাঁড়াতাম তখন আমরা আমাদের দু’হাত দিয়ে দুজনকে ধরতাম। একজনের ডানহাত পরবর্তী জনের বামহাত ধরত, তার ডান হাত আবার তার পরের জনের বামহাত। এভাবে একটা হিউম্যান চেইন গঠন করতাম আমরা। এই চেইনে প্রতিটি মানুষকে এক-একটি নোড হিসেবে কল্পনা করতে পারি আমরা। সেই নোডের ডাটা আইটেম আমাদের মূলদেহ আর ডানহাত হল পয়েন্টার। কি এখনো সহজ মনে হচ্ছে না? তাহলে একটা ছবি দেখা যাক।



এতক্ষণ আমরা যে হাত ধরাধরির কাহিনী বর্ণনা করছিলাম, এই চিত্রটা আসলে তারই মানসচিত্র (ভিজুয়ালাইজেশন - visualization)। প্রথম নোড থেকে মূলত সিঙ্গলি লিংকড লিস্টের অগ্রযাত্রা শুরু হয়। প্রথম নোডের ডাটা আইটেম অংশে প্রথম ভ্যালুটা থাকে। প্রথম নোডের পয়েন্টার অংশ দ্বিতীয় নোডকে নির্দেশ করে। এভাবে চলতে থাকে। একটা লিস্টের পরিসমাপ্তিকে মূলত নাল (Null) রেফারেন্স দ্বারা চিহ্নিত করা হয়। অনেক সময় None পরিসমাপ্তি বুঝানো হয়। পরিসমাপ্তি দ্বারা বুঝানো হচ্ছে যে এই নোডের পরে আর কোন নোড নেই।

#### অপারেশন

এক নজরে আমরা এখন সিঙ্গলি লিংকড লিস্টের সকল অপারেশন দেখব। সাধারণত নয় ধরনের ফাংশন বা মেথডের মাধ্যমে এই অপারেশনগুলো সম্পাদিত হয়।

#### appendleft(item)

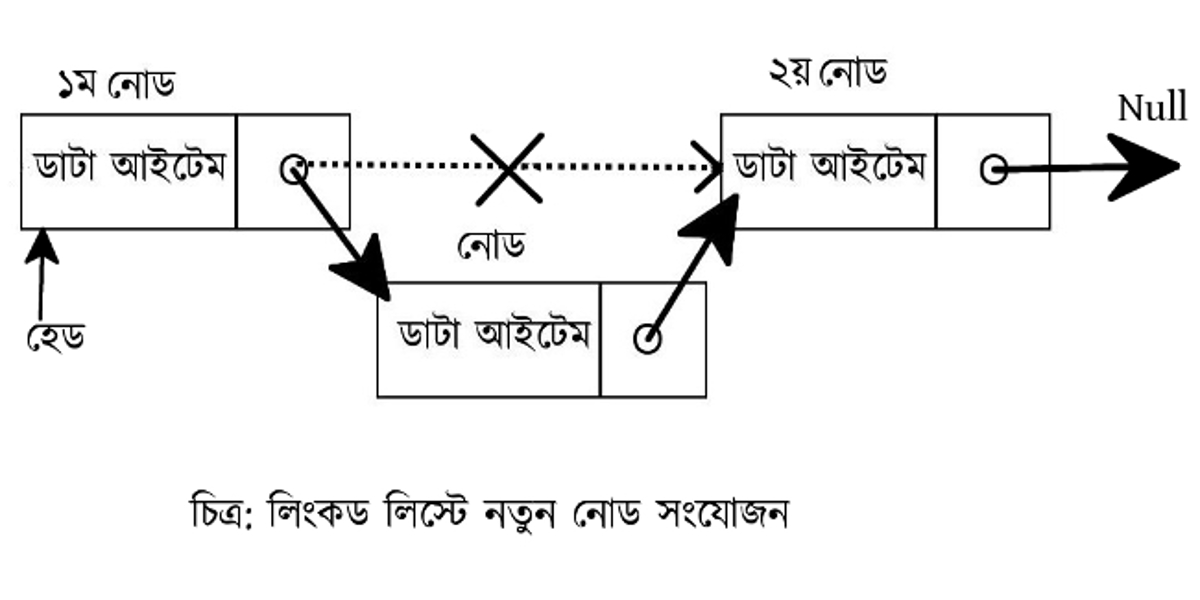
লিস্টের শুরুতে নতুন একটি নোড সংযোজনের মাধ্যমে নতুন কোন ডাটা আইটেম সংযোজন করার ক্ষেত্রে এই ফাংশন বা মেথডটি ব্যবহার করা হয়। আর্গুমেন্ট বা প্যারামিটার হিসেবে আলোচ্য ডাটা আইটেমটিকে গ্রহণ করলেও এটি কোন কিছু রিটার্ন করে না।

#### append(item)

লিস্টের শেষে নতুন একটি নোড সংযোজনের মাধ্যমে নতুন কোন ডাটা আইটেম সংযোজন করার ক্ষেত্রে এই ফাংশন বা মেথডটি ব্যবহার করা হয়। আর্গুমেন্ট বা প্যারামিটার হিসেবে আলোচ্য ডাটা আইটেমটিকে গ্রহণ করলেও এটি কোন কিছু রিটার্ন করে না।

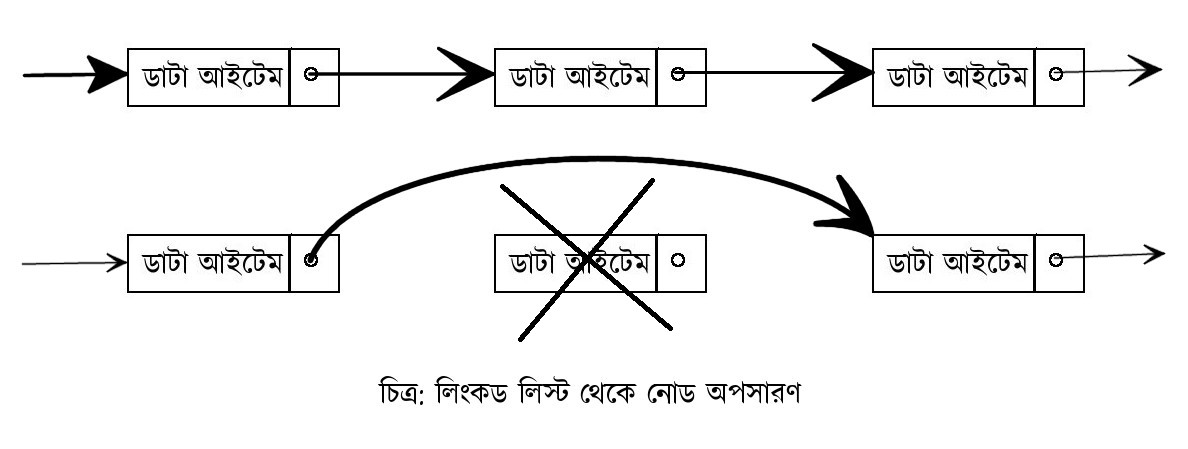
#### insert(position, item)

লিস্টের নির্দিষ্ট কোন অবস্থানে (পজিশানে) নতুন একটি নোড সংযোজনের মাধ্যমে নতুন কোন ডাটা আইটেম সংযোজন করার ক্ষেত্রে এই ফাংশন বা মেথডটি ব্যবহার করা হয়। আর্গুমেন্ট বা প্যারামিটার হিসেবে আলোচ্য পজিশান ও ডাটা আইটেমটিকে গ্রহণ করলেও এটি কোন কিছু রিটার্ন করে না।



#### remove(item)

লিস্টের শুরু থেকে নির্দিষ্ট কোন ডাটা আইটেম ও সংশ্লিষ্ট নোড অপসারণ করার জন্য এই ফাংশন বা মেথডটি ব্যবহার করা হয়। আর্গুমেন্ট বা প্যারামিটার হিসেবে আইটেমটিকে গ্রহণ করলেও এটি কোন কিছু রিটার্ন করে না। (ধরে নেয়া হয়, আইটেমটি লিস্টে রয়েছে)



#### pop()

সাধারণত এই ফাংশন বা মেথড লিস্টের শেষের ডাটা আইটেমটিকে ও সংশ্লিষ্ট নোড অপসারণ করে এবং ডাটা আইটেম রিটার্ন করে। (ধরে নেয়া হয়, লিস্টে অন্ততপক্ষে একটি আইটেম রয়েছে)

#### is\_empty()

এটি একটি বুলিয়ান ফাংশন (বা মেথড)। লিস্ট খালি কিনা সেটি চেক করে True বা False রিটার্ন করে। এর কোন প্যারামিটার নেই।

#### size()

এই ফাংশন (বা মেথড) লিংকড লিস্টের মোট আইটেম (নাকি নোড?) সংখ্যা রিটার্ন করে। এরও কোন প্যারামিটার নেই।

#### search(item)

এটি একটি বুলিয়ান ফাংশন (বা মেথড)। একটি আইটেমকে আর্গুমেন্ট বা প্যারামিটার হিসেবে গ্রহণ করে লিংকড লিস্টে সেটি রয়েছে কিনা তা সার্চ করে দেখে এবং True বা False রিটার্ন করে।

#### index(item)

এই ফাংশন বা মেথডটি একটি আইটেমকে আর্গুমেন্ট বা প্যারামিটার হিসেবে গ্রহণ করে। তারপর সেটিকে লিংকড লিস্টে সার্চ করে দেখে এর পজিশন রিটার্ন করে। (ধরে নেয়া হয়, আইটেমটি লিস্টে রয়েছে)

#### printlist()

এই ফাংশন বা মেথডটি লিস্টের সবগুলো আইটেমকে প্রিন্ট করবে। (ধরে নেয়া হয়, লিস্টে ন্যূনতম একটি আইটেম রয়েছে।)

#### ইমপ্লিমেন্টেশন

স্টাক, কিউ বা ডেকের মত সিঙ্গলি লিংকড লিস্টের ইমপ্লিমেন্টেশন সরল প্রকৃতির নয়। খানিকটা জটিল হলেও পাইথনে (3.x) ভাল ধারণা থাকলে ইমপ্লিমেন্ট করতে কষ্ট হবার কথা নয়। আসলে ডাটা স্ট্রাকচার বা অ্যালগরিদম দেখে ইমপ্লিমেন্ট করার মত কোন বিষয় নয়। কি হচ্ছে বা কি হবে তা সম্পর্কে স্পষ্ট ধারণা থাকলে সহজেই কোড লিখে ফেলা যায়। তারপরও [উদাহরণ হিসেবে এটা](https://github.com/maateen/data-structure-practice/blob/master/LinkedList/SinglyLinkedList.py) দেখা যেতে পারে।

অদূর ভবিষ্যতে আমরা ডাবলি (Doubly) লিংকড লিস্ট ও [সার্কুলার (Circular) লিংকড লিস্ট](https://bn.maateen.me/data-structure-circular-linked-list/) সম্পর্কেও জানব। তবে তার আগে সিঙ্গলি লিংকড লিস্টে হাত পাকাতে হবে আমাদের। সেজন্য সিঙ্গলি লিংকড লিস্ট ব্যবহার করে আমরা স্টাক ও কিউ ইমপ্লিমেন্ট করব। আর [হ্যাকারর‍্যাংক](https://www.hackerrank.com/domains/data-structures/linked-lists) ও [হ্যাকারআর্থ](https://www.hackerearth.com/practice/data-structures/linked-list/singly-linked-list/)-এর কিছু প্রব্লেম

### ডাটা স্ট্রাকচার : সার্কুলার লিংকড লিস্ট

আগস্ট 5, 2017 তারিখে, বিভাগ: [প্রোগ্রামিং](https://blog.maateen.me/bn/categories/%e0%a6%aa%e0%a7%8d%e0%a6%b0%e0%a7%8b%e0%a6%97%e0%a7%8d%e0%a6%b0%e0%a6%be%e0%a6%ae%e0%a6%bf%e0%a6%82), [পাইথন](https://blog.maateen.me/bn/categories/%e0%a6%aa%e0%a6%be%e0%a6%87%e0%a6%a5%e0%a6%a8), [ডাটা স্ট্রাকচার](https://blog.maateen.me/bn/categories/%e0%a6%a1%e0%a6%be%e0%a6%9f%e0%a6%be-%e0%a6%b8%e0%a7%8d%e0%a6%9f%e0%a7%8d%e0%a6%b0%e0%a6%be%e0%a6%95%e0%a6%9a%e0%a6%be%e0%a6%b0)

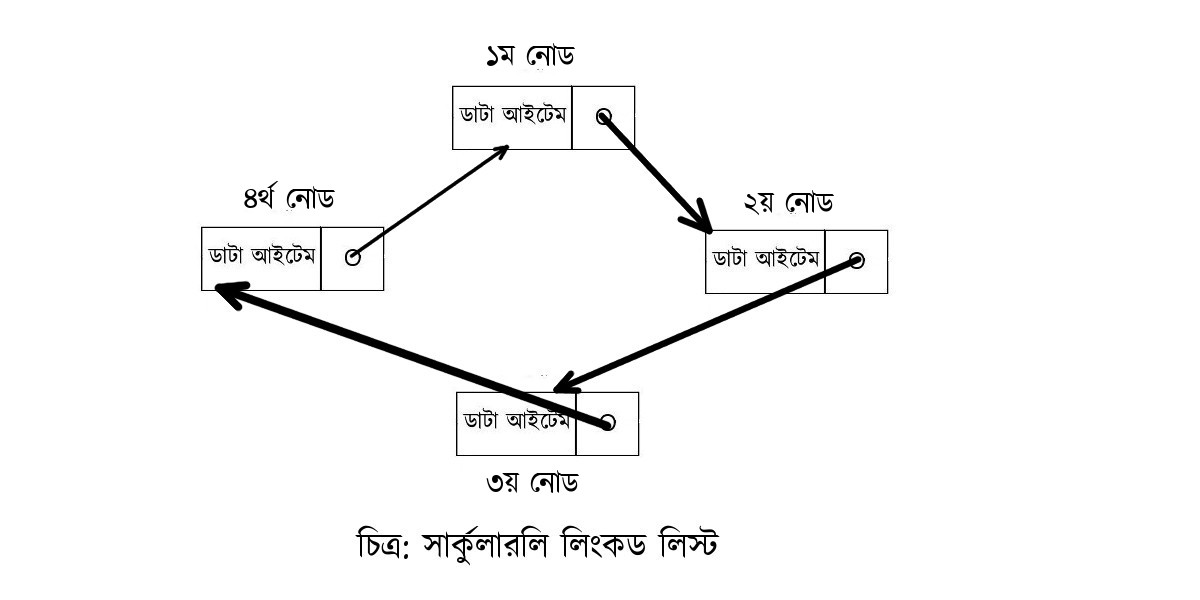
[সিঙ্গলি লিংকড লিস্ট](https://bn.maateen.me/data-structure-singly-linked-list/) সম্পর্কে পরিষ্কার ধারণা থাকলে সার্কুলার লিংকড লিস্ট বুঝতে পারাটা একেবারে পানির মত সহজ। তবে ভুলে গেলেও কোন সমস্যা নেই। আমরা এমনিতেই ফ্লাশব্যাকে যাব।

[সিঙ্গলি লিংকড লিস্ট](https://bn.maateen.me/data-structure-singly-linked-list/) হল কতগুলো নোডের চেইন বা সমাহার। প্রতিটি নোডে দুটি ফিল্ড থাকে। প্রথম ফিল্ডে থাকে ডাটা আইটেম আর শেষের ফিল্ডে থাকে পয়েন্টার বা পরবর্তী নোডের লিংক বা রেফারেন্স। নোড চেইনের পরিসমাপ্তিকে মূলত নাল (Null) রেফারেন্স দ্বারা চিহ্নিত করা হয়। অনেক সময় None দিয়ে পরিসমাপ্তি বুঝানো হয়। পরিসমাপ্তি দ্বারা বুঝানো হচ্ছে যে এই নোডের পরে আর কোন নোড নেই।

সিঙ্গলি লিংকড লিস্টের সাথে সার্কুলার লিংকড লিস্টের পার্থক্য মূলত এই নাল (Null) রেফারেন্সে। সার্কুলার লিংকড লিস্টে কোন নাল রেফারেন্স থাকে না। বরং এর শেষ নোডে প্রথম নোডের লিংক বা রেফারেন্স থাকে। ফলশ্রুতিতে সার্কুলার লিংকড লিস্টকে চক্রাকার বেনজিন বলয়ের সাথে তুলনা করা যায়।

স্কুলের সেই দুরন্তপনার দিনগুলোয় ফেরত যাওয়া যাক। আমাদের কি ক্রীড়া প্রতিযোগীতার বিভিন্ন ইভেন্টের কথা মনে আছে? মনে আছে কি সেই মোরগ লড়াইয়ের কথা? আমরা অনেকে হাত ধরাধরি করে চক্রাকারে দাঁড়াতাম আর তার মাঝখানে প্রতিযোগীতা হত। আমাদের বন্ধুরা মোরগের মত লাফালাফি করতে করতে একে অপরকে ধাক্কা দিত। কেউ পড়ে গেলেই সে আউট। তখন চক্রটাকে একটু ছোট করে আনা হত। মানসপট থেকে সেই দিনগুলো কি হারিয়ে গেছে?

এবার এই হাত ধরাধরি করে চক্রাকারে দাঁড়ানোটাকে বিবেচনা করা যাক। এই হিউম্যান চেইনে প্রত্যেক হিউম্যানকে নোড হিসেবে কল্পনা করা যায়। আমাদের মূলদেহকে সেই নোডের ডাটা আইটেম আর ডানহাতকে পরবর্তী নোডের লিংক বা রেফারেন্স হিসেবে কল্পনা করা যায়। আর বামহাত? কথায় আছে ভাল কাজে বাম হাত দিতে নাই। যাহোক, এবার একটা ছবি দেখা যাক।



এতক্ষণ আমরা যে হাত ধরাধরির কাহিনী বর্ণনা করছিলাম, এই চিত্রটা আসলে তারই মানসচিত্র (ভিজুয়ালাইজেশন – visualization)। ধরা যাক, প্রথম নোড থেকে সার্কুলার লিংকড লিস্টের অগ্রযাত্রা শুরু হয়েছে। প্রথম নোডের ডাটা আইটেম অংশে প্রথম ভ্যালুটা থাকে। প্রথম নোডের পয়েন্টার অংশ দ্বিতীয় নোডকে নির্দেশ করে। এভাবে চলতে থাকে। আর চতুর্থ নোডের পয়েন্টার অংশ প্রথম নোডকে নির্দেশ করে। এটাই হল চক্রাকার লিংকড লিস্ট।

#### অপারেশন

এক নজরে আমরা এখন সার্কুলার লিংকড লিস্টের সকল অপারেশন দেখব। সাধারণত দশ ধরনের ফাংশন বা মেথডের মাধ্যমে এই অপারেশনগুলো সম্পাদিত হয়।

#### appendleft(item)

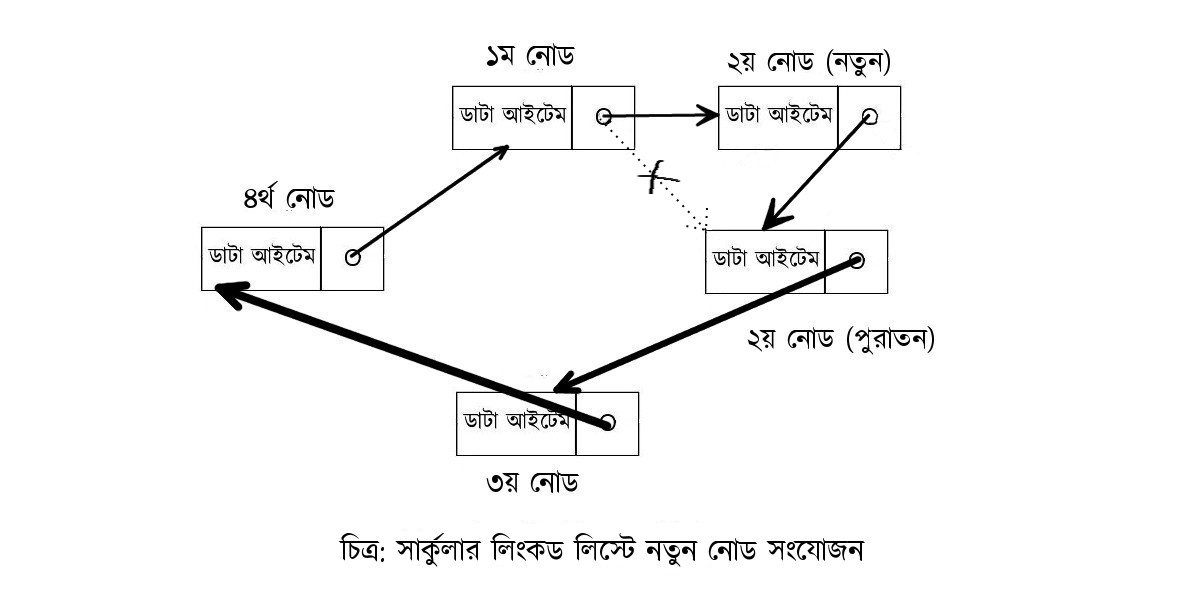
লিস্টের শুরুতে নতুন একটি নোড সংযোজনের মাধ্যমে নতুন কোন ডাটা আইটেম সংযোজন করার ক্ষেত্রে এই ফাংশন বা মেথডটি ব্যবহার করা হয়। আর্গুমেন্ট বা প্যারামিটার হিসেবে আলোচ্য ডাটা আইটেমটিকে গ্রহণ করলেও এটি কোন কিছু রিটার্ন করে না।

#### append(item)

লিস্টের শেষে নতুন একটি নোড সংযোজনের মাধ্যমে নতুন কোন ডাটা আইটেম সংযোজন করার ক্ষেত্রে এই ফাংশন বা মেথডটি ব্যবহার করা হয়। আর্গুমেন্ট বা প্যারামিটার হিসেবে আলোচ্য ডাটা আইটেমটিকে গ্রহণ করলেও এটি কোন কিছু রিটার্ন করে না।

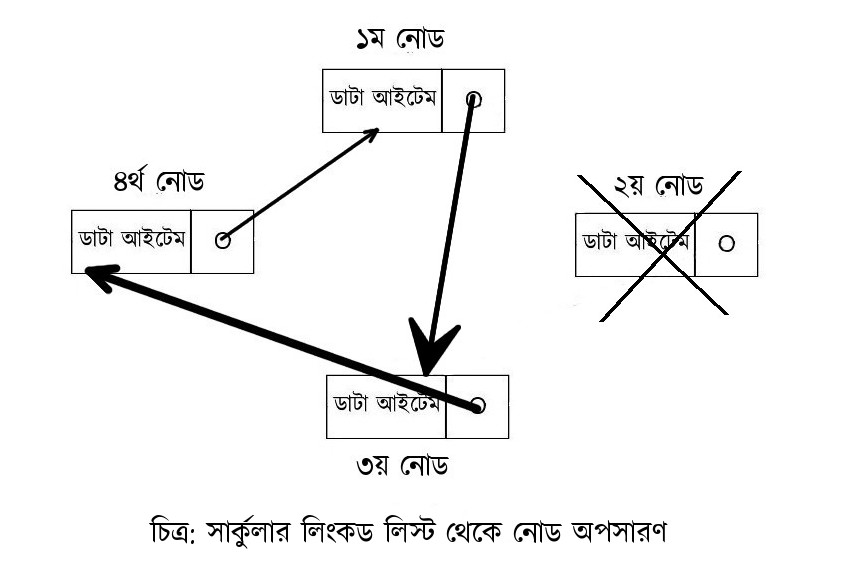
#### insert(position, item)

লিস্টের নির্দিষ্ট কোন অবস্থানে (পজিশানে) নতুন একটি নোড সংযোজনের মাধ্যমে নতুন কোন ডাটা আইটেম সংযোজন করার ক্ষেত্রে এই ফাংশন বা মেথডটি ব্যবহার করা হয়। আর্গুমেন্ট বা প্যারামিটার হিসেবে আলোচ্য পজিশান ও ডাটা আইটেমটিকে গ্রহণ করলেও এটি কোন কিছু রিটার্ন করে না।



#### remove(item)

লিস্টের শুরু থেকে নির্দিষ্ট কোন ডাটা আইটেম ও সংশ্লিষ্ট নোড অপসারণ করার জন্য এই ফাংশন বা মেথডটি ব্যবহার করা হয়। আর্গুমেন্ট বা প্যারামিটার হিসেবে আইটেমটিকে গ্রহণ করলেও এটি কোন কিছু রিটার্ন করে না। (ধরে নেয়া হয়, আইটেমটি লিস্টে রয়েছে)



#### popleft()

সাধারণত এই ফাংশন বা মেথড লিস্টের প্রথম ডাটা আইটেমটিকে ও সংশ্লিষ্ট নোড অপসারণ করে এবং ডাটা আইটেম রিটার্ন করে। (ধরে নেয়া হয়, লিস্টে অন্ততপক্ষে একটি আইটেম রয়েছে)

#### pop()

সাধারণত এই ফাংশন বা মেথড লিস্টের শেষের ডাটা আইটেমটিকে ও সংশ্লিষ্ট নোড অপসারণ করে এবং ডাটা আইটেম রিটার্ন করে। (ধরে নেয়া হয়, লিস্টে অন্ততপক্ষে একটি আইটেম রয়েছে)

#### is\_empty()

এটি একটি বুলিয়ান ফাংশন (বা মেথড)। লিস্ট খালি কিনা সেটি চেক করে True বা False রিটার্ন করে। এর কোন প্যারামিটার নেই।

#### size()

এই ফাংশন (বা মেথড) লিংকড লিস্টের মোট আইটেম (নাকি নোড?) সংখ্যা রিটার্ন করে। এরও কোন প্যারামিটার নেই।

#### search(item)

এটি একটি বুলিয়ান ফাংশন (বা মেথড)। একটি আইটেমকে আর্গুমেন্ট বা প্যারামিটার হিসেবে গ্রহণ করে লিংকড লিস্টে সেটি রয়েছে কিনা তা সার্চ করে দেখে এবং True বা False রিটার্ন করে।

#### index(item)

এই ফাংশন বা মেথডটি একটি আইটেমকে আর্গুমেন্ট বা প্যারামিটার হিসেবে গ্রহণ করে। তারপর সেটিকে লিংকড লিস্টে সার্চ করে দেখে এর পজিশন রিটার্ন করে। (ধরে নেয়া হয়, আইটেমটি লিস্টে রয়েছে)

#### printlist()

এই ফাংশন বা মেথডটি লিস্টের সবগুলো আইটেমকে প্রিন্ট করবে। (ধরে নেয়া হয়, লিস্টে ন্যূনতম একটি আইটেম রয়েছে।)

#### ইমপ্লিমেন্টেশন

আমরা এখন জানি যে, সার্কুলার লিংকড লিস্ট হল কতগুলো নোডের চেইন। প্রতিটি নোডকে ধারণ করার জন্য একটা Node() ক্লাস তৈরি করতে হবে আমাদের।

class Node():

def \_\_init\_\_(self, item=None, next\_node=None):

self.item = item

self.next\_node = next\_node

এই ক্লাসের আর্গুমেন্ট দুটি: item ও next\_node। item দিয়ে ডাটা আইটেমকে বুঝানো হচ্ছে। আর next\_node মূলত একটি পয়েন্টার যাতে পরবর্তী নোডের রেফারেন্স জমা থাকবে। প্রাথমিকভাবে আর্গুমেন্ট দুটির ভ্যালু None।

এবার সার্কুলার লিংকড লিস্টের অপারেশনসমূহ করার জন্য দশটা মেথড তৈরি করব আমরা। এই মেথডগুলো থাকবে CircularLinkedList() ক্লাসের অধীন।

class CircularLinkedList():

def \_\_init\_\_(self, head=None):

self.head = head

এই ক্লাসের আর্গুমেন্ট কেবলমাত্র একটি: head। হেড লিস্টের প্রথম নোডকে নির্দেশ করে। প্রাথমিকভাবে head এর ভ্যালু None।

প্রথমে appendleft(item) মেথড নিয়ে ভাবা যাক।

def appendleft(self, item):

new\_node = Node(item)

new\_node.next\_node = self.head

current = self.head

if self.head is not None:

while current.next\_node is not self.head:

current = current.next\_node

current.next\_node = new\_node

else:

new\_node.next\_node = new\_node

self.head = new\_node

এই মেথড সবসময় হেডে আইটেম (আসলে নোড) যোগ করবে। তাই প্রথমেই নোড ক্লাসের মাধ্যমে একটি নতুন নোড তৈরি করেছি আমরা। লিস্টে নতুন হেড আসা মানে বর্তমান হেডকে চলে যেতে হবে দ্বিতীয় অবস্থানে। তাই নতুন নোডের next\_node ভ্যারিয়েবলে self.head (ক্লাস ভ্যারিয়েবল)-এ থাকা নোডের রেফারেন্স দিয়েছি আমরা। এখন ঘটনা হল, কোন লিস্টে self.head এ কিছু থাকতেও পারে আবার নাও থাকতে পারে। কিছু না থাকার মানে হল লিস্টে কেবলমাত্র আমাদের যোগ করা নোডটিই আছে। সেক্ষত্রে এই নোড তার next\_node ভ্যারিয়েবলে নিজেই নিজেকে পয়েন্ট করবে। আর self.head এ কোন নোড থাকলে ব্যাপারটা বেশ গোলমেলে। সেক্ষেত্রে লিস্টের সর্বশেষ নোড আমাদের এই নতুন নোডটিকে পয়েন্ট করবে। তবেই না হবে সার্কুলার লিংকড লিস্ট।

এবার আমরা append(item) মেথডটি নিয়ে ভাবব। এই মেথড সবসময় লিস্টের শেষে নতুন আইটেম (আসলে নোড) যোগ করবে। তাহলে কিন্তু ব্যাপারটা বেশ সহজ। একটা নতুন নোড তৈরি করতে হবে। বর্তমানে লিস্টের শেষে থাকা নোডটি এই নতুন নোডটিকে পয়েন্ট করবে। আর নতুন নোডটি পয়েন্ট করবে হেডকে।

def append(self, item):

new\_node = Node(item)

current = self.head

if current:

while True:

if current.next\_node is self.head:

current.next\_node = new\_node

new\_node.next\_node = self.head

break

else:

current = current.next\_node

else:

self.head = new\_node

new\_node.next\_node = new\_node

insert(position, item) মেথডটি বেশ মজার। লিস্টের হেডে (জিরো পজিশনে) বা লিস্টের একেবারে শেষের পজিশানে যোগ করার ক্ষেত্রে যথাক্রমে appendleft(item) ও append(item) মেথড কল করে কাজ সারতে পারব। কিন্তু দুটি নোডের মাঝখানে কোন নোড যোগ করতে চাইলে একটু কষ্ট করতে হবে আমাদের।

def insert(self, position, item):

if position == 0:

self.appendleft(item)

print(item, "inserted to position", position)

elif position == self.size():

self.append(item)

print(item, "inserted to position", position)

elif position > self.size():

print("Index out of range")

else:

current = self.head

index = 0

while current:

if index != position:

previous = current

current = current.next\_node

index += 1

else:

new\_node = Node(item, current)

previous.next\_node = new\_node

current = False

print(item, "inserted to position", position)

ধরা যাক, চার নম্বর পজিশনে একটা নতুন নোড যোগ করব আমরা। তারমানে বর্তমানে চার নম্বর পজিশনে থাকা নোডটি চলে যাবে পাঁচ নম্বর পজিশনে, পাঁচ নম্বর পজিশনে থাকা নোডটি চলে যাবে ছয় নম্বর পজিশনে। অর্থাৎ এক ঘর করে সরবে। তাই নতুন একটি নোড তৈরি করে এর next\_node ভ্যারিয়েবল দিয়ে বর্তমানে চার নম্বর পজিশনে থাকা নোডটিকে পয়েন্ট করাব আমরা। আর তিন নম্বর পজিশনে থাকা নোডটির next\_node ভ্যারিয়েবল পয়েন্ট করবে নতুন নোডটিকে। ব্যাস, হয়ে গেল ইনসার্ট।

is\_empty() মেথডটা বেশ সহজ। লিস্টে যদি কমপক্ষে একটি নোডও থাকে তবে self.head এ তার রেফারেন্স থাকবেই। শুধু হেড খালি (None) কিনা সেটা চেক করে দেখলেই চলে। খালি হলে True রিটার্ন করবে, অন্যথায় False রিটার্ন করবে।

def is\_empty(self):

if self.head == None:

return True

else:

return False

size() মেথডটি is\_empty() মেথডের মতই সহজ। ছোটবেলার মত করে একটা while লুপ ঘুরিয়ে নোড কাউন্ট করা শুরু করতে হবে। আর যখনই কোন নোডের next\_node এর রেফারেন্সে হেডকে পাওয়া যাবে তখনই লুপ থামিয়ে দিতে হবে। এবার মোট কতবার লুপ ঘুরল সেটা হিসেব করলেই কিন্তু লিস্টের সাইজ পাওয়া যাবে। তাই না?

def size(self):

current = self.head

count = 0

while current:

count += 1

if current.next\_node is self.head:

current = False

else:

current = current.next\_node

return count

উপরের size() মেথডের ইমপ্লিমেন্টেশন ঠিকমত বুঝে থাকলে index(item) মেথডটি ইমপ্লিমেন্ট করা সহজ হবে। আমরা জানি যে, এই মেথডের কাজ হল কোন আইটেমের ইনডেক্স পজিশন রিটার্ন করা। আমরা ধরে নিলাম, সার্কুলার লিংকড লিস্টের ইনডেক্স সিস্টেম পাইথনের চিরাচরিত লিস্টের মতই। মানে জিরো থেকে শুরু হবে।

def index(self, item):

current = self.head

index = 0

while current:

if current.item == item:

return index

elif current.next\_node is self.head:

break

else:

current = current.next\_node

index += 1

return None

এজন্য আমরা হেড থেকে আইটেম খোঁজা শুরু করেছি। না পেলে পরবর্তী নোডে চলে গিয়েছি। আর যখনই পরবর্তী নোডে চলে গিয়েছি তখনই ইনডেক্সের মান ১ করে বাড়িয়ে দিয়েছি। যখন কাঙ্ক্ষিত ডাটা আইটেম কোন নোডে পাওয়া গিয়েছে তখন লুপ টার্মিনেট করে দিয়েছি। এভাবে আমরা আমাদের কাঙ্ক্ষিত ডাটা আইটেমের (আসলে নোডের) ইনডেক্স নম্বর পেয়ে গিয়েছি।

search(item) মেথড আগের index(item) মেথডের মতই। পার্থক্য শুধু এতটুকু যে এই মেথড কাঙ্ক্ষিত আইটেম খুঁজে পেলে True রিটার্ন করবে। অন্যথায় False রিটার্ন করবে।

def search(self, item):

current = self.head

found = False

while current and not found:

if current.item == item:

found = True

elif current.next\_node is self.head:

current = False

else:

current = current.next\_node

if current is None:

print("Item not found")

return found

popleft() মেথডের কাজ হল appendleft(item) মেথডের ঠিক উল্টো। মানে, লিস্টের প্রথম নোডটিকে রিমুভ করা ও এর আইটেমকে রিটার্ন করা।

def popleft(self):

if self.is\_empty():

print("Empty list")

else:

current = self.head

temp = current.item

if current.next\_node is self.head:

self.head = None

else:

self.head = current.next\_node

next\_node = self.head

while next\_node.next\_node is not current:

next\_node = next\_node.next\_node

next\_node.next\_node = self.head

del current

return temp

self.head-এ যেহেতু লিস্টের প্রথম নোডটি থাকে, তাই এটা রিমুভ করব আমরা। এটা রিমুভ করার পর এর ঠিক পরের নোডটি হেডে চলে আসবে। আর লিস্টের সর্বশেষ নোডের next\_node ভ্যারিয়েবল এই নতুন হেডকে পয়েন্ট করবে। সর্বশেষ নোডটা বের করার জন্য রিমুভ করার পর যে নোডটি হেড হবে সেটি থেকে লুপ ঘুরানো শুরু করেছি আমরা। আমরা দেখতে চেয়েছি কোন নোডের next\_node ভ্যারিয়েবল রিমুভেবল নোডটিকে পয়েন্ট করে। যেটি পয়েন্ট করে সেটিই সর্বশেষ নোড। এবার প্রথম নোডটি রিমুভ করার পালা। রিমুভ করার আগে নোডের আইটেমটা একটা temp ভ্যারিয়েবলে রেখে দিয়েছি যাতে নোড রিমুভ করার পরে আইটেমটা রিটার্ন করা যায়।

pop() মেথডের কাজ append(item) মেথডের উল্টো। মানে, লিস্টের সর্বশেষ নোডটিকে রিমুভ করা ও এর আইটেমকে রিটার্ন করা।

def pop(self):

if self.is\_empty():

print("Empty list")

else:

current = self.head

previous = None

while current.next\_node is not self.head:

previous = current

current = current.next\_node

if current == self.head:

self.head = None

else:

previous.next\_node = self.head

temp = current.item

del current

return temp

আমাদের প্রথম কাজ হল সর্বশেষ নোডটিকে খুঁজে বের করা। কিভাবে করা যায়? আমরা কিন্তু ইতিমধ্যে শিখে এসেছি। যেই নোডের next\_node ভ্যারিয়েবল হেডকে পয়েন্ট করে সেটাই সর্বশেষ নোড। সেজন্য একটা লুপ ঘুরিয়ে আমরা সর্বশেষ নোডটিকে খুঁজে বের করেছি। রিমুভ করার আগে নোডের আইটেমটা একটা temp ভ্যারিয়েবলে রেখে দিয়েছি যাতে নোড রিমুভ করার পরে আইটেমটা রিটার্ন করা যায়।

remove(item) মেথডটি একটু কঠিন মনে হতে পারে। তাই একটু মনোযোগী হতে হবে এবার। (এতক্ষণ অমনোযোগী ছিলাম আমরা?)

def remove(self, item):

if self.is\_empty():

print("Empty list")

else:

current = self.head

previous = None

found = False

while current and not found:

if current.item == item:

found = True

elif current.next\_node is self.head:

current = None

else:

previous = current

current = current.next\_node

if current is None:

print("Item not found")

elif previous is None:

self.popleft()

print(item, "removed")

else:

temp = current.next\_node

del current

print(item, "removed")

previous.next\_node = temp

যেহেতু আমরা কোন আইটেম রিমুভ করব, তাই সবার আগে চেক করে দেখা দরকার লিস্টে আদৌ কোন নোড আছে কিনা। সেজন্য আমরা আমাদের is\_empty() মেথডের সাহায্য নিয়েছি। লিস্ট খালি না হলে যে আইটেমটি রিমুভ করতে হবে একটা while লুপ ঘুরিয়ে সেটাকে খোঁজ দ্যা সার্চ করেছি। কাঙ্ক্ষিত ডাটা আইটেম হেডে থাকলে popleft() মেথড কল করাই যথেষ্ট। অন্যথায়, যেই নোডে কাঙ্ক্ষিত ডাটা আইটেমটি রয়েছে সেটাকে কারেন্ট নোড হিসেবে ধরে নিলাম। এবার এখানে কারসাজি রয়েছে। কারেন্ট নোডের next\_node ভ্যারিয়েবল যে নোডটিকে পয়েন্ট করছে তার রেফারেন্স কারেন্ট নোডের ঠিক আগের নোডের next\_node ভ্যারিয়েবলে অ্যাসাইন করে দিয়েছি। তারপর কারেন্ট নোডকে ডিলিট করে দিয়েছি আমরা। আচ্ছা, del current এই স্টেটমেন্টের সহীহ ফযিলত কি? বলুন তো!

printlist() মেথডটা আসলে অমন আহামরি কঠিন কিছু না। বিখ্যাত নাবিক ক্রিস্টোফার কলম্বাসের মত আবিষ্কারের নেশায় মেতে উঠতে হবে। একটা জাহাজে চড়ে থুক্কু লুপ ঘুরিয়ে সবগুলো নোড আবিষ্কার করতে হবে। তারপর সেই নোডের ডাটা আইটেমকে প্রিন্ট করতে হবে।

def printlist(self):

if self.is\_empty():

print("Empty list")

else:

current = self.head

print(current.item)

while current.next\_node:

current = current.next\_node

if current is self.head:

break

else:

print(current.item)

উপরের আলোচনা থেকে এটা স্পষ্ট বোঝাই যাচ্ছে, সার্কুলার লিংকড লিস্টের অপারেশন ও ইমপ্লিমেন্টেশন অনেকটা সিঙ্গলি লিংকড লিস্টের মতই। একটু চেষ্টা করলেই পারা যাবে। তারপরও [উদাহরণ হিসেবে এটা](https://github.com/maateen/data-structure-practice/blob/master/LinkedList/CircularLinkedList.py)

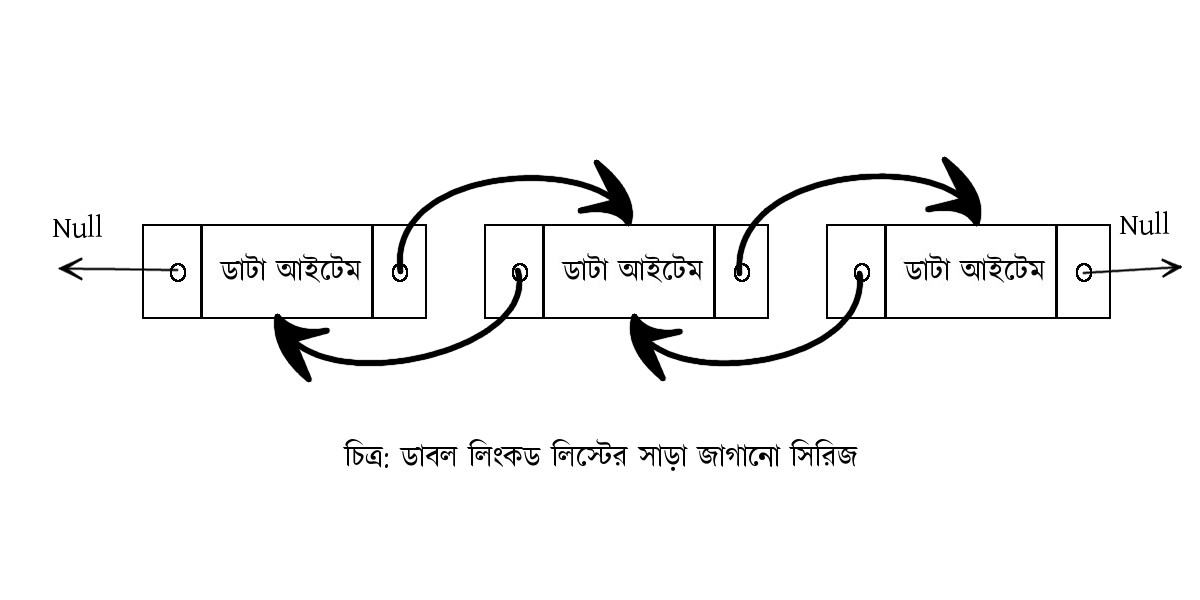
### ডাটা স্ট্রাকচার : ডাবলি লিংকড লিস্ট

আগস্ট 8, 2017 তারিখে, বিভাগ: [প্রোগ্রামিং](https://blog.maateen.me/bn/categories/%e0%a6%aa%e0%a7%8d%e0%a6%b0%e0%a7%8b%e0%a6%97%e0%a7%8d%e0%a6%b0%e0%a6%be%e0%a6%ae%e0%a6%bf%e0%a6%82), [পাইথন](https://blog.maateen.me/bn/categories/%e0%a6%aa%e0%a6%be%e0%a6%87%e0%a6%a5%e0%a6%a8), [ডাটা স্ট্রাকচার](https://blog.maateen.me/bn/categories/%e0%a6%a1%e0%a6%be%e0%a6%9f%e0%a6%be-%e0%a6%b8%e0%a7%8d%e0%a6%9f%e0%a7%8d%e0%a6%b0%e0%a6%be%e0%a6%95%e0%a6%9a%e0%a6%be%e0%a6%b0)

ইতিমধ্যে আমরা [সিঙ্গলি লিংকড লিস্ট](https://bn.maateen.me/data-structure-singly-linked-list/) ও [সার্কুলার লিংকড লিস্ট](https://bn.maateen.me/data-structure-circular-linked-list/) সম্পর্কে অল্প-বিস্তর ধারণা লাভ করার চেষ্টা করেছি। তারই ধারাবাহিকতায় এখন ডাবলি লিংকড লিস্ট সম্পর্কে জানব।

সিঙ্গলি লিংকড লিস্টের মত ডাবলি লিংকড লিস্টও কতগুলো নোডের চেইন বা সমাহার। কিন্তু পার্থক্য হল নোডের ফিল্ড সংখ্যায়। সিঙ্গলি লিংকড লিস্টে প্রতিটি নোডে দুটি ফিল্ড থাকে। কিন্তু ডাবলি লিংকড লিস্টে প্রতিটি নোডে তিনটি ফিল্ড থাকে। প্রথম ফিল্ডে থাকে প্রিভিয়াস (previous) পয়েন্টার বা পূর্ববর্তী নোডের লিংক (রেফারেন্স), মাঝের ফিল্ডে থাকে ডাটা আইটেম আর শেষের ফিল্ডে থাকে নেক্সট পয়েন্টার বা পরবর্তী নোডের লিংক (রেফারেন্স)। যেহেতু প্রতিটি নোডে দুটি করে পয়েন্টার থাকে, তাই এই নোড চেইনেরও দুটি প্রান্ত থাকে। দুই প্রান্তেই নোড চেইনের পরিসমাপ্তিকে মূলত নাল (Null) রেফারেন্স দ্বারা চিহ্নিত করা হয়। অনেক সময় None দিয়েও পরিসমাপ্তি বুঝানো হয়। পরিসমাপ্তি দ্বারা বুঝানো হচ্ছে যে, এই নোডের পরে আর কোন নোড নেই।

ব্যাপারটা সহজে বোঝার জন্য ধরা যাক, আমরা দশ-পনেরজন ভাই-ব্রাদার হাত ধরাধরি করে ডিএসএলআরের সামনে পোজ দিচ্ছি। এই যে হাত ধরাধরি করে দাঁড়ালাম, এটাকে হিউম্যান চেইন বলা যায়। এই হিউম্যান চেইনে প্রত্যেক হিউম্যানকে নোড হিসেবে কল্পনা করা যায়। আমাদের বাম হাতকে প্রিভিয়াস পয়েন্টার, মূলদেহকে সেই নোডের ডাটা আইটেম আর ডানহাতকে নেক্সট পয়েন্টার হিসেবে কল্পনা করা যায়। এতে চেইনে দুই প্রান্তের দুইজনের একটা করে হাত খালি থাকবে। নতুন কেউ এই চেইনে যুক্ত হতে চাইলে হাত ধরে দাঁড়িয়ে গেলেই হবে। এখনও ঘোলাটে মনে হচ্ছে? তাহলে একটা চিত্র দেখা যাক।



এতক্ষণ আমরা যে হাত ধরাধরির কাহিনী বর্ণনা করছিলাম, এই চিত্রটা আসলে তারই মানসচিত্র (ভিজুয়ালাইজেশন – visualization)। ধরা যাক, প্রথম নোড থেকে ডাবলি লিংকড লিস্টের অগ্রযাত্রা শুরু হয়েছে। প্রথম নোডের প্রিভিয়াস পয়েন্টার নাল (Null), কারণ এর আগে কোন নোড নেই। এই নোডের ডাটা আইটেম অংশে প্রথম ভ্যালুটা থাকে। প্রথম নোডের নেক্সট পয়েন্টার অংশ দ্বিতীয় নোডকে নির্দেশ করে। এভাবে চলতে থাকে। আর তৃতীয় নোডের নেক্সট পয়েন্টার নাল (Null), কারণ এর পরে কোন নোড নেই। এটাই হল ডাবলি লিংকড লিস্ট।

#### অপারেশন

এক নজরে আমরা এখন ডাবলি লিংকড লিস্টের সকল অপারেশন দেখব। সাধারণত দশ ধরনের ফাংশন বা মেথডের মাধ্যমে এই অপারেশনগুলো সম্পাদিত হয়।

#### appendleft(item)

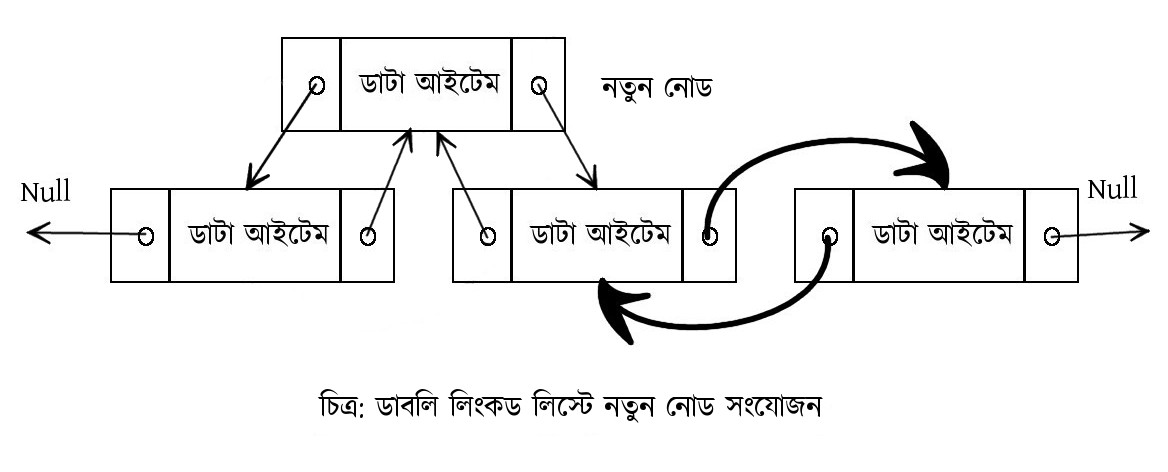
লিস্টের শুরুতে নতুন একটি নোড সংযোজনের মাধ্যমে নতুন কোন ডাটা আইটেম সংযোজন করার ক্ষেত্রে এই ফাংশন বা মেথডটি ব্যবহার করা হয়। আর্গুমেন্ট বা প্যারামিটার হিসেবে আলোচ্য ডাটা আইটেমটিকে গ্রহণ করলেও এটি কোন কিছু রিটার্ন করে না।

#### append(item)

লিস্টের শেষে নতুন একটি নোড সংযোজনের মাধ্যমে নতুন কোন ডাটা আইটেম সংযোজন করার ক্ষেত্রে এই ফাংশন বা মেথডটি ব্যবহার করা হয়। আর্গুমেন্ট বা প্যারামিটার হিসেবে আলোচ্য ডাটা আইটেমটিকে গ্রহণ করলেও এটি কোন কিছু রিটার্ন করে না।

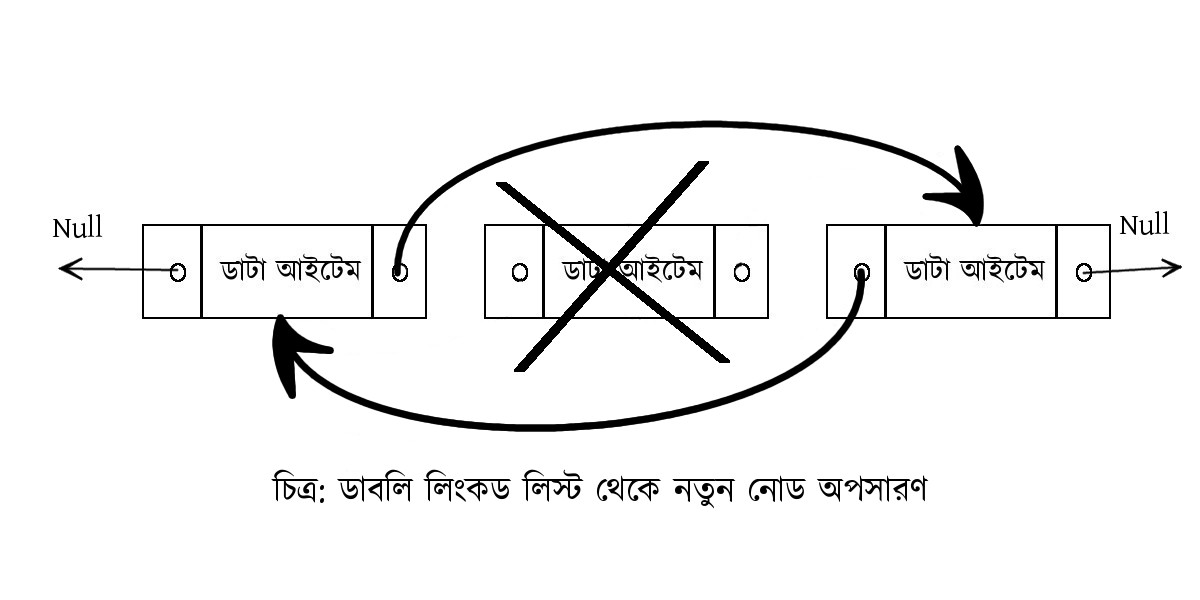
#### insert(position, item)

লিস্টের নির্দিষ্ট কোন অবস্থানে (পজিশানে) নতুন একটি নোড সংযোজনের মাধ্যমে নতুন কোন ডাটা আইটেম সংযোজন করার ক্ষেত্রে এই ফাংশন বা মেথডটি ব্যবহার করা হয়। আর্গুমেন্ট বা প্যারামিটার হিসেবে আলোচ্য পজিশান ও ডাটা আইটেমটিকে গ্রহণ করলেও এটি কোন কিছু রিটার্ন করে না।



#### remove(item)

লিস্টের শুরু থেকে নির্দিষ্ট কোন ডাটা আইটেম ও সংশ্লিষ্ট নোড অপসারণ করার জন্য এই ফাংশন বা মেথডটি ব্যবহার করা হয়। আর্গুমেন্ট বা প্যারামিটার হিসেবে আইটেমটিকে গ্রহণ করলেও এটি কোন কিছু রিটার্ন করে না। (ধরে নেয়া হয়, আইটেমটি লিস্টে রয়েছে)



#### popleft()

সাধারণত এই ফাংশন বা মেথড লিস্টের প্রথম ডাটা আইটেমটিকে ও সংশ্লিষ্ট নোড অপসারণ করে এবং ডাটা আইটেম রিটার্ন করে। (ধরে নেয়া হয়, লিস্টে অন্ততপক্ষে একটি আইটেম রয়েছে)

#### pop()

সাধারণত এই ফাংশন বা মেথড লিস্টের শেষের ডাটা আইটেমটিকে ও সংশ্লিষ্ট নোড অপসারণ করে এবং ডাটা আইটেম রিটার্ন করে। (ধরে নেয়া হয়, লিস্টে অন্ততপক্ষে একটি আইটেম রয়েছে)

#### is\_empty()

এটি একটি বুলিয়ান ফাংশন (বা মেথড)। লিস্ট খালি কিনা সেটি চেক করে True বা False রিটার্ন করে। এর কোন প্যারামিটার নেই।

#### size()

এই ফাংশন (বা মেথড) লিংকড লিস্টের মোট আইটেম (নাকি নোড?) সংখ্যা রিটার্ন করে। এরও কোন প্যারামিটার নেই।

#### search(item)

এটি একটি বুলিয়ান ফাংশন (বা মেথড)। একটি আইটেমকে আর্গুমেন্ট বা প্যারামিটার হিসেবে গ্রহণ করে লিংকড লিস্টে সেটি রয়েছে কিনা তা সার্চ করে দেখে এবং True বা False রিটার্ন করে।

#### index(item)

এই ফাংশন বা মেথডটি একটি আইটেমকে আর্গুমেন্ট বা প্যারামিটার হিসেবে গ্রহণ করে। তারপর সেটিকে লিংকড লিস্টে সার্চ করে দেখে এর পজিশন রিটার্ন করে। (ধরে নেয়া হয়, আইটেমটি লিস্টে রয়েছে)

#### printlist()

এই ফাংশন বা মেথডটি লিস্টের সবগুলো আইটেমকে প্রিন্ট করবে। (ধরে নেয়া হয়, লিস্টে ন্যূনতম একটি আইটেম রয়েছে।)

#### ইমপ্লিমেন্টেশন

আমরা এখন জানি যে, ডাবলি লিংকড লিস্ট হল কতগুলো নোডের চেইন। প্রতিটি নোডকে ধারণ করার জন্য একটা Node() ক্লাস তৈরি করতে হবে আমাদের।

class Node():

def \_\_init\_\_(self, item=None, previous\_node=None, next\_node=None):

self.item = item

self.previous\_node = previous\_node

self.next\_node = next\_node

এই ক্লাসের আর্গুমেন্ট তিনটি: item, previous\_node ও next\_node। item দিয়ে ডাটা আইটেমকে বুঝানো হচ্ছে। আর previous\_node হল উপরে উল্লেখিত পূর্ববর্তী নোডের রেফারেন্স এবং next\_node হল পরবর্তী নোডের রেফারেন্স। প্রাথমিকভাবে আর্গুমেন্ট তিনটির ভ্যালু None।

এবার ডাবলি লিংকড লিস্টের অপারেশনসমূহ করার জন্য দশটা মেথড তৈরি করব আমরা। এই মেথডগুলো থাকবে DoublyLinkedList() ক্লাসের অধীন।

class DoublyLinkedList():

def \_\_init\_\_(self, head=None, tail=None):

self.head = head

self.tail = tail

এই ক্লাসের আর্গুমেন্ট দুটি: head ও tail। হেড লিস্টের প্রথম নোডকে নির্দেশ করে। আর tail নির্দেশ করে লিস্টের সর্বশেষ নোডকে। প্রাথমিকভাবে এদের ভ্যালু None।

প্রথমে appendleft(item) মেথড নিয়ে ভাবা যাক।

def appendleft(self, item):

new\_node = Node(item)

new\_node.previous\_node = None

new\_node.next\_node = self.head

if self.head is None:

self.tail = new\_node

else:

self.head.previous\_node = new\_node

self.head = new\_node

এই মেথড সবসময় হেডে আইটেম (আসলে নোড) যোগ করবে। তাই প্রথমেই নোড ক্লাসের মাধ্যমে একটি নতুন নোড তৈরি করেছি আমরা। লিস্টে নতুন হেড আসা মানে বর্তমান হেডকে চলে যেতে হবে দ্বিতীয় অবস্থানে। তাই নতুন নোডের next\_node ভ্যারিয়েবলে self.head (ক্লাস ভ্যারিয়েবল)-এ থাকা নোডের রেফারেন্স দিয়েছি আমরা। আর এই নতুন নোডের আগে যেহেতু কোন নোড নেই তাই এর previous\_node ভ্যারিয়েবলে None অ্যাসাইন করে দিয়েছি। এখন ঘটনা হল, কোন লিস্টে self.head এ কিছু থাকতেও পারে আবার নাও থাকতে পারে। কিছু না থাকার মানে হল লিস্টে কেবলমাত্র আমাদের যোগ করা নোডটিই আছে। সেক্ষত্রে এই নোডটিই লিস্টের প্রথম ও শেষ নোড। তাই self.tail এ আমরা নতুন নোডের রেফারেন্স দিয়েছি। আর self.head এ কোন নোড থাকলে তার previous\_node ভ্যারিয়েবলে নতুন নোডের রেফারেন্স দিতে হবে। সবশেষে, self.head (ক্লাস ভ্যারিয়েবল)-এ আমাদের নতুন নোডের রেফারেন্স দিয়েছি।

এবার আমরা append(item) মেথডটি নিয়ে ভাবব। এই মেথড সবসময় লিস্টের শেষে নতুন আইটেম (আসলে নোড) যোগ করবে। তাহলে কিন্তু ব্যাপারটা বেশ সহজ। একটা নতুন নোড তৈরি করতে হবে। বর্তমানে লিস্টের শেষে (self.tail এ) থাকা নোডটির next\_node ভ্যারিয়েবল এই নতুন নোডটিকে পয়েন্ট করবে। আর নতুন নোডটির previous\_node ভ্যারিয়েবল পয়েন্ট করবে self.tail এ থাকা নোডটি।

def append(self, item):

new\_node = Node(item)

if self.head is None:

self.head = self.tail = new\_node

else:

new\_node.previous\_node = self.tail

new\_node.next\_node = None

self.tail.next\_node = new\_node

self.tail = new\_node

insert(position, item) মেথডটি বেশ মজার। লিস্টের হেডে (জিরো পজিশনে) বা লিস্টের একেবারে শেষের পজিশনে (টেইলে) যোগ করার ক্ষেত্রে যথাক্রমে appendleft(item) ও append(item) মেথড কল করে কাজ সারতে পারব। কিন্তু দুটি নোডের মাঝখানে কোন নোড যোগ করতে চাইলে একটু কষ্ট করতে হবে আমাদের।

def insert(self, position, item):

if position == 0:

self.appendleft(item)

print(item, "inserted to position", position)

elif position == self.size():

self.append(item)

print(item, "inserted to position", position)

elif position > self.size():

print("Index out of range")

else:

current = self.head

index = 0

while current:

if index != position:

previous = current

current = current.next\_node

index += 1

else:

new\_node = Node(item, previous, current)

previous.next\_node = new\_node

current.previous\_node = new\_node

current = False

print(item, "inserted to position", position)

ধরা যাক, চার নম্বর পজিশনে একটা নতুন নোড যোগ করব আমরা। তারমানে বর্তমানে চার নম্বর পজিশনে থাকা নোডটি চলে যাবে পাঁচ নম্বর পজিশনে, পাঁচ নম্বর পজিশনে থাকা নোডটি চলে যাবে ছয় নম্বর পজিশনে। অর্থাৎ এক ঘর করে সরবে। তাই নতুন একটি নোড তৈরি করে এর next\_node ভ্যারিয়েবল দিয়ে বর্তমানে চার নম্বর পজিশনে থাকা নোডটিকে ও previous\_node ভ্যারিয়েবল দিয়ে বর্তমানে তিন নম্বর পজিশনে থাকা নোডটিকে পয়েন্ট করাব আমরা। আর তিন নম্বর পজিশনে থাকা নোডটির next\_node ভ্যারিয়েবল পয়েন্ট করবে নতুন নোডটিকে। ব্যাস, হয়ে গেল ইনসার্ট।

is\_empty() মেথডটা বেশ সহজ। লিস্টে যদি কমপক্ষে একটি নোডও থাকে তবে self.head এ তার রেফারেন্স থাকবেই। শুধু হেড খালি (None) কিনা সেটা চেক করে দেখলেই চলে। খালি হলে True রিটার্ন করবে, অন্যথায় False রিটার্ন করবে।

def is\_empty(self):

if self.head is None:

return True

else:

return False

size() মেথডটি is\_empty() মেথডের মতই সহজ। ছোটবেলার মত করে একটা while লুপ ঘুরিয়ে নোড কাউন্ট করা শুরু করতে হবে। আর যখনই কোন নোডের next\_node এর রেফারেন্সে None পাওয়া যাবে তখনই লুপ থামিয়ে দিতে হবে। এবার মোট কতবার লুপ ঘুরল সেটা হিসেব করলেই কিন্তু লিস্টের সাইজ পাওয়া যাবে। তাই না?

def size(self):

current = self.head

count = 0

while current:

count += 1

current = current.next\_node

return count

উপরের size() মেথডের ইমপ্লিমেন্টেশন ঠিকমত বুঝে থাকলে index(item) মেথডটি ইমপ্লিমেন্ট করা সহজ হবে। আমরা জানি যে, এই মেথডের কাজ হল কোন আইটেমের ইনডেক্স পজিশন রিটার্ন করা। আমরা ধরে নিলাম, ডাবলি লিংকড লিস্টের ইনডেক্স সিস্টেম পাইথনের চিরাচরিত লিস্টের মতই। মানে জিরো থেকে শুরু হবে।

def index(self, item):

current = self.head

index = 0

while current:

if current.item == item:

return index

else:

current = current.next\_node

index += 1

return None

এজন্য আমরা হেড থেকে আইটেম খোঁজা শুরু করেছি। না পেলে পরবর্তী নোডে চলে গিয়েছি। আর যখনই পরবর্তী নোডে চলে গিয়েছি তখনই ইনডেক্সের মান ১ করে বাড়িয়ে দিয়েছি। যখন কাঙ্ক্ষিত ডাটা আইটেম কোন নোডে পাওয়া গিয়েছে তখন লুপ টার্মিনেট করে দিয়েছি। এভাবে আমরা আমাদের কাঙ্ক্ষিত ডাটা আইটেমের (আসলে নোডের) ইনডেক্স নম্বর পেয়ে গিয়েছি।

search(item) মেথড আগের index(item) মেথডের মতই। পার্থক্য শুধু এতটুকু যে এই মেথড কাঙ্ক্ষিত আইটেম খুঁজে পেলে True রিটার্ন করবে। অন্যথায় False রিটার্ন করবে।

def search(self, item):

current = self.head

found = False

while current and not found:

if current.item == item:

found = True

else:

current = current.next\_node

if current is None:

print("Item not found")

return found

popleft() মেথডের কাজ হল appendleft(item) মেথডের ঠিক উল্টো। মানে, লিস্টের প্রথম নোডটিকে রিমুভ করা ও এর আইটেমকে রিটার্ন করা।

def popleft(self):

if self.is\_empty():

print("Empty list")

else:

current = self.head

next\_node = current.next\_node

if next\_node is None:

temp = current.item

del current

self.head = self.tail = None

return temp

else:

temp = current.item

del current

next\_node.previous\_node = None

self.head = next\_node

return temp

self.head-এ যেহেতু লিস্টের প্রথম নোডটি থাকে, তাই এটা রিমুভ করব আমরা। এটা রিমুভ করার পর এর ঠিক পরের নোডটি হেডে চলে আসবে। আর নতুন হেডের previous\_node ভ্যারিয়েবল কোন নোডকে পয়েন্ট করবে না, তাই None অ্যাসাইন করব। এবার প্রথম নোডটি রিমুভ করার পালা। রিমুভ করার আগে নোডের আইটেমটা একটা temp ভ্যারিয়েবলে রেখে দিয়েছি যাতে নোড রিমুভ করার পরে আইটেমটা রিটার্ন করা যায়। উল্লেখ্য যে, লিস্টে কেবলমাত্র একটি নোড থাকলে, এটি রিমুভ করার পর self.head ও self.tail এ None অ্যাসাইন করে দেব আমরা। বলুন তো কেন?

pop() মেথডের কাজ append(item) মেথডের উল্টো। মানে, লিস্টের সর্বশেষ নোডটিকে রিমুভ করা ও এর আইটেমকে রিটার্ন করা।

def pop(self):

if self.is\_empty():

print("Empty list")

else:

current = self.tail

previous = current.previous\_node

if previous is None:

temp = current.item

del current

self.head = self.tail = None

return temp

else:

temp = current.item

del current

previous.next\_node = None

self.tail = previous

return temp

self.tail-এ লিস্টের সর্বশেষ নোডটিকে পাব আমরা। এই নোডের ঠিক আগের নোডটি হবে নতুন টেইল। তাই এর next\_node ভ্যারিয়েবলে None অ্যাসাইন করে দিয়েছি। সর্বশেষ নোডটি রিমুভ করার আগে নোডের আইটেমটা একটা temp ভ্যারিয়েবলে রেখে দিয়েছি যাতে নোড রিমুভ করার পরে আইটেমটা রিটার্ন করা যায়।

remove(item) মেথডটি একটু কঠিন মনে হতে পারে। তাই একটু মনোযোগী হতে হবে এবার। (এতক্ষণ অমনোযোগী ছিলাম আমরা?)

def remove(self, item):

if self.is\_empty():

print("Empty list")

else:

current = self.head

previous = None

found = False

while current and not found:

if current.item == item:

found = True

else:

previous = current

current = current.next\_node

if current is None:

print("Item not found")

elif previous is None:

self.popleft()

print(item, "removed")

else:

temp = current.next\_node

del current

print(item, "removed")

previous.next\_node = temp

temp.previous\_node = previous

যেহেতু আমরা কোন আইটেম রিমুভ করব, তাই সবার আগে চেক করে দেখা দরকার লিস্টে আদৌ কোন নোড আছে কিনা। সেজন্য আমরা আমাদের is\_empty() মেথডের সাহায্য নিয়েছি। লিস্ট খালি না হলে যে আইটেমটি রিমুভ করতে হবে একটা while লুপ ঘুরিয়ে সেটাকে খোঁজ দ্যা সার্চ করেছি। কাঙ্ক্ষিত ডাটা আইটেম হেডে থাকলে popleft() আর টেইলে থাকলে pop() মেথড কল করাই যথেষ্ট। অন্যথায়, যেই নোডে কাঙ্ক্ষিত ডাটা আইটেমটি রয়েছে সেটাকে কারেন্ট নোড হিসেবে ধরে নিলাম। এবার এখানে কারসাজি রয়েছে। কারেন্ট নোডের next\_node ভ্যারিয়েবল যে নোডটিকে পয়েন্ট করছে তার রেফারেন্স কারেন্ট নোডের ঠিক আগের নোডের next\_node ভ্যারিয়েবলে অ্যাসাইন করে দিয়েছি। আর কারেন্ট নোডের previous\_node ভ্যারিয়েবল যে নোডটিকে পয়েন্ট করছে তার রেফারেন্স কারেন্ট নোডের ঠিক পরের নোডের previous\_node ভ্যারিয়েবলে অ্যাসাইন করে দিয়েছি। তারপর কারেন্ট নোডকে ডিলিট করে দিয়েছি আমরা। আচ্ছা, del current এই স্টেটমেন্টের সহীহ ফযিলত কি? বলুন তো!

printlist() মেথডটা আসলে অমন আহামরি কঠিন কিছু না। বিখ্যাত নাবিক ক্রিস্টোফার কলম্বাসের মত আবিষ্কারের নেশায় মেতে উঠতে হবে। একটা জাহাজে চড়ে থুক্কু লুপ ঘুরিয়ে সবগুলো নোড আবিষ্কার করতে হবে। তারপর সেই নোডের ডাটা আইটেমকে প্রিন্ট করতে হবে।

def printlist(self):

if self.is\_empty():

print("Empty list")

else:

current = self.head

while current:

print(current.item)

current = current.next\_node

আশা করি একটু চেষ্টা করলেই ডাবলি লিংকড লিস্ট ইমপ্লিমেন্ট করতে পারা যাবে এখন। তারপরও উদাহরণ হিসেবে এটা দেখে যেতে পারেন।

#### ওয়েব প্রোগ্রামিং

ওয়েব প্রোগ্রামিংকে আমরা দুই ভাগে ভাগ করতে পারি— প্রথমটি হচ্ছে, ফ্রন্টেন্ড (Frontend) বা ক্লায়েন্ট সাইড প্রোগ্রামিং এবং দ্বিতীয়টি ব্যাকেন্ড (Backend) বা সার্ভার সাইড প্রোগ্রামিং।

একটা ওয়েবসাইট বা ওয়েব অ্যাপের যে অংশটুকু আমরা চোখে দেখতে পারি বা ইউজার যার সাথে ইন্টার-অ্যাক্ট করে তাই ফ্রন্টেন্ড আর যা দেখতে পারি না অথবা যা ইউজারের বিভিন্ন রিকোয়েস্টে রেসপন্স করে তা ব্যাকেন্ড। ফ্রন্টেন্ডে আমরা HTML5, CSS3, Javascript-এর মতো ল্যাঙ্গুয়েজগুলো ব্যবহার করতে পারি। এই তিনটির মধ্যে Javascript ক্লায়েন্ট সাইড প্রোগ্রামিং ল্যাঙ্গুয়েজ হলেও HTML5 ও CSS3 কোনো প্রোগ্রামিং ল্যাঙ্গুয়েজ নয়; বরং এগুলো হলো মার্ক-আপ ল্যাঙ্গুয়েজ। যেমন- XML একটা মার্ক-আপ ল্যাঙ্গুয়েজ। ক্লায়েন্ট সাইড প্রোগ্রামিং ল্যাঙ্গুয়েজগুলো সাধারণত ইউজারের ব্রাউজারে রান করে।

সার্ভার সাইড ল্যাঙ্গুয়েজগুলোর মধ্যে রয়েছে PHP, Ruby, Python, Go, C#, Visual Basic, Java, Perl ইত্যাদি। NodeJS এর কল্যাণে এখন অবশ্য জাভাস্ক্রিপ্টও সার্ভারে রান করানো যায়। নোডজেএস হলো ক্রোমের V8 ইঞ্জিনের ওপর নির্মিত জাভাস্ক্রিপ্ট রানটাইম। তবে আমাদের আলোচনার টপিক, পাইথন নিয়ে।

পাইথন দিয়ে খুব সহজেই আমরা ওয়েবসাইট বা ওয়েব অ্যাপ তৈরি করতে পারি। আচ্ছা, ওয়েবসাইট ও ওয়েব অ্যাপের মধ্যে পার্থক্য কী? প্রাথমিকভাবে ওয়েবসাইট তথ্যমূলক হয়। আর ওয়েব অ্যাপে ইউজার বিভিন্ন অ্যাকশন পারফর্ম করতে পারে। অন্য কথায়, ওয়েবসাইট হলো সুপারসেট এবং ওয়েব অ্যাপ হলো ওয়েবসাইটের একটি সাবসেট। একটি ওয়েবসাইট একটি ওয়েব অ্যাপকে ধারণ করতে পারে। যাহোক, সাধারণ পাইথন কোড লিখেই আমরা ওয়েব সাইট বা ওয়েব অ্যাপ তৈরি করতে পারি। কিন্তু ডেভেলপাররা বা প্রোগ্রামাররা তাদের কাজকে সহজ ও গতিশীল করার জন্য বিভিন্ন ওয়েব ফ্রেমওয়ার্ক ব্যবহার করে থাকে।

এখন প্রশ্ন হলো, ওয়েব ফ্রেমওয়ার্ক কী? পাইথনের ভাষ্যমতে-

ওয়েব ফ্রেমওয়ার্ক এমন একটা কোড লাইব্রেরি যা রিলায়াবল, স্কালাবল ও মেইনটেইনেবল ওয়েব অ্যাপ্লিকেশন তৈরি করার ক্ষেত্রে একজন ডেভেলপারের লাইফকে সহজতর করে তোলে।

মূলত ওয়েব ফ্রেমওয়ার্ক ডেভেলপারদের অনেকগুলো API অফার করে যার মাধ্যমে কাস্টম কোড লিখে খুব সহজেই ওয়েব অ্যাপ্লিকেশন তৈরি করা যায়। একটা ওয়েব অ্যাপ্লিকেশনে URL ম্যাপি (রাউটিং), টেমপ্লেটিং, ডেটাবেস ম্যানিপুলেশন, ইউজার ম্যানেজমেন্ট (রেজিস্ট্রেশন ও লগইন) এবং সেশন স্টোরেজ কমনলি থাকে। এসব কাজের জন্য একজন ডেভেলপারকে প্রতিনিয়ত একই কোড বারবার লিখতে হয়। আর ফ্রেমওয়ার্ক হলো এসব ঝামেলা থেকে মুক্তির আশ্বাস। ফ্রেমওয়ার্কের নীতি হলো ‘Less code, more work’। ফ্রেমওয়ার্ক ব্যবহার করে আমরা ওপরের বিষয়গুলো অনেক সহজে হ্যান্ডেল করতে পারি।

পাইথনের অনেকগুলো ওয়েব ফ্রেমওয়ার্ক রয়েছে। কোনোটাই কোনোটার চেয়ে কম না। কিন্তু তারপরও কিছু ফ্রেমওয়ার্ক এদের স্বকীয়তার কারণে হয়ে উঠেছে অনন্য। এগুলো হলো-

* Django (জ্যাঙ্গো), ওয়েবসাইট : <https://www.djangoproject.com/>
* Flask (ফ্লাস্ক), ওয়েবসাইট : <http://flask.pocoo.org/>
* Bottle (বটল), ওয়েবসাইট : <http://bottlepy.org/docs/dev/>
* WEB2PY (ওয়েবটুপাই), ওয়েবসাইট : <http://www.web2py.com/>
* Falcon (ফ্যালকন), ওয়েবসাইট: <https://falconframework.org/>
* Pyramid (পিরামিড), ওয়েবসাইট : <https://trypyramid.com/>
* TurboGears (টার্বোগিয়ারস), ওয়েবসাইট : <http://www.turbogears.org/>
* web.py (ওয়েবপাই), ওয়েবসাইট: <http://webpy.org/>

এ ছাড়া আরও অনেক ফ্রেমওয়ার্ক রয়েছে। কিন্তু প্রশ্ন হলো এত এত ফ্রেমওয়ার্কের ভিড়ে আমরা শিখব কোনটা?

এতগুলো ফ্রেমওয়ার্কের ভেতরে সবচেয়ে জনপ্রিয় ফ্রেমওয়ার্ক দুটি হলো জ্যাঙ্গো ও ফ্লাস্ক। সুতরাং আমাদের এই দুটির মধ্যে যেকোনো একটি বেছে নিতে হবে। কিন্তু এ ক্ষেত্রে ভাবার মতো অনেক বিষয় আছে। প্রথমেই ভাবা যাক চাকরির কথা। মোটামুটি সব পাইথননির্ভর ওয়েব ডেভেলপমেন্ট কোম্পানিই তাদের নিত্যদিনের কাজে জ্যাঙ্গো ব্যবহার করে। মোটামুটি সব লোকাল বা ইন্টারন্যাশনাল জব পোস্টই (পাইথন ওয়েব প্রোগ্রামিং সংক্রান্ত) কিন্তু জ্যাঙ্গোকেন্দ্রিক। তবে ফ্লাস্ক নিয়েও অনেক কোম্পানি কাজ করে। কিন্তু ফ্লাস্কে তারা হালকাপাতলা কিছু রেস্টফুল এপিআই ইমপ্লিমেন্ট করা বাদে আর সব কাজ জ্যাঙ্গোতেই করে। এসব দিক বিবেচনা করলে জ্যাঙ্গো শেখাই অধিক যুক্তিসংগত।

এবার ভাবা যাক ফ্রেমওয়ার্ক দুটির উদ্দেশ্য ও দৃষ্টিভঙ্গির কথা। ফ্লাস্ক একটি মাইক্রো-ফ্রেমওয়ার্ক। মাইক্রো মানে ফ্লাস্কের কোর (Core)-কে যথাসম্ভব সিম্পল কিন্তু এক্সটেনসিবল রাখা হয়েছে। ফ্লাস্ক আমাদের জন্য নিজে কোনো ডিসিশন নেবে না (যেমন : কোনো ডেটাবেস ব্যবহার করতে হবে)। অন্যদিকে জ্যাঙ্গো হলো, ব্যাটারিজ-ইনক্লুডেড (batteries-included) ফ্রেমওয়ার্ক। ওয়েব ডেভেলপমেন্টের বেশির ভাগ ঝামেলাই জ্যাঙ্গো হ্যান্ডেল করে। তাই আমরা শুধু নিজেদের অ্যাপ তৈরিতে পূর্ণ মনোযোগ দিতে পারি। ফ্লাস্ক দিয়ে আমরা যদি একটা বড় প্রজেক্ট করতে চাই তখন আমাদের অনেকগুলো এক্সটেনশন ও থার্ড-পার্টি মডিউল ব্যবহার করতে হবে এবং নিজেদের অনেক ঝামেলা পোহাতে হবে। কিন্তু জ্যাঙ্গো আমাদের সবকিছু একটি প্যাকেজের মধ্যেই দিচ্ছে। অথেনটিকেশন, কন্টেন্ট ম্যানেজমেন্ট, সাইট ম্যাপ, আরএসএস ফিল্ড- সবকিছুই জ্যাঙ্গো হ্যান্ডেল করতে পারে। তা ছাড়া জ্যাঙ্গোর প্রচুর থার্ড-পার্টি প্যাকেজ রয়েছে, যা জ্যাঙ্গোকে করে তুলেছে অনন্য।

আমি নিজে যখন জ্যাঙ্গো শেখা শুরু করেছিলাম, তখন এর আর্কিটেকচার আমার খুব কঠিন মনে হতো। MTV (মডেল-ভিউ-টেমপ্লেট) ডিজাইন প্যাটার্ন আমার কাছে অনেক কনফিউজিং মনে হতো। তাই দিন শেষে আমার কাছে জ্যাঙ্গো ভয়াবহ রূপ ধারণ করে। সে জন্য আমি ফ্লাস্কে মুভ করি। ফ্লাস্ক মাইক্রো-ফ্রেমওয়ার্ক ও খুব সহজ হওয়ায় অল্পতেই ওয়েব ফ্রেমওয়ার্কের মৌলিক বিষয়গুলো শিখতে পারি। এরপর MTV ডিজাইন প্যাটার্ন অনুসরণ করে একটা URL শর্টেনার ও ব্লগ ইঞ্জিন তৈরি করি। তারপরে এক শুভদিনে আবার ফিরে আসি জ্যাঙ্গোর কাছে। কিন্তু অবাক করা বিষয় হলো, এবার জ্যাঙ্গোকে ‘জলবৎ তরলং’ অর্থাৎ পানির মতো সহজ মন হতে লাগল। ফলে অল্প দিনেই জ্যাঙ্গোতে হাত পেকে গেল।

এবার আশা করি আপনাদের সবার সিদ্ধান্ত নিতে সুবিধা হবে। তবে আমি বলব, জ্যাঙ্গো ও ফ্লাস্ক দুটোই শেখা উচিত। কারণ, ছোটখাটো প্রজেক্টে জ্যাঙ্গো ব্যবহার করা অনেকটা মশা মারতে কামান দাগার মতো। ওয়েবে স্ট্যাটিক সাইট জেনারেটর নামে আরেকটি ধারণা আছে। স্ট্যাটিক সাইট জেনারেটরগুলো সিম্পল HTML ফাইল তৈরি করে।

পাইথনের নামকরা কয়েকটি স্ট্যাটিক সাইট জেনারেটর হলো-

* Pelican (পেলিকান): <http://docs.getpelican.com/en/stable/>
* Lektor (লেক্টর): <https://www.getlektor.com/>
* Nikola (নিকোলা), ওয়েবসাইট : <https://getnikola.com/>

এবার চলুন, ফ্লাস্কের অফিশিয়াল ডক থেকে দুই-চার লাইন শিখি। প্রথমেই ফ্লাস্ক ইনস্টল করার জন্য টার্মিনালে কমান্ড দেব-

sudo pip3 install Flask

এবার hello.py নামে একটা ফাইল তৈরি করে তাতে নিচের মতো কোড লিখব-

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route("/")

def hello():

return "Hello World!"

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

এবার এই স্ক্রিপ্ট বা ফাইলটিকে চালানোর জন্য টার্মিনালে কমান্ড দেব-

python3 hello.py

দেখতে পাব টার্মিনালে \* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit) এ রকম একটি লেখা উঠেছে। তার মানে আমরা সাকসেসফুল। এই অবস্থায় টার্মিনাল বন্ধ না করেই ব্রাউজারটি ওপেন করব। এবার ব্রাউজারের অ্যাড্রেস বারে http://localhost:5000/ লিখে এন্টার চাপব। ব্যস! সাথে সাথে Hello World! লেখাটা দেখতে পাব।

ফ্লাস্কের সাথে ডেটাবেস ব্যবহার করার জন্য আমরা Flask-SQLAlchemy এক্সটেনশনটি ব্যবহার করতে পারি। এটি হলো SQLAlchemy নামের একটি বিখ্যাত পাইথন SQL টুলকিটের ফ্লাস্ক বাইন্ডিং। SQLAlchemy ব্যবহার করে আমরা যেকোনো পাইথন স্ক্রিপ্টেই ডেটাবেস ব্যবহার করতে পারি। যাহোক, প্রথমেই এটি ইনস্টল করে নেওয়া যাক।

sudo pip3 install Flask-SQLAlchemy

আমাদের পুরোনো সেই hello.py-তে এবার নিচের মতো করে কোড লেখা যাক-

import os

from flask import Flask

from flask\_sqlalchemy import SQLAlchemy

app = Flask(\_\_name\_\_)

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///test.db'

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = True

db = SQLAlchemy(app)

if not os.path.exists('test.db'):

db.create\_all()

@app.route("/")

def hello():

return"Hello World!"

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

এবার ফাইলটিকে রান করলে দেখব hello.py এর সাথে একই ডিরেক্টরিতে test.db নামে নতুন একটি ফাইল তৈরি হয়েছে। আসলে এটি হলো ডেটাবেস। ওপরের প্রোগ্রামে আমরা ডেটাবেস হিসেবে SQLite-এর ব্যবহার করেছি। SQLite হলো একটি সহজ-সাধারণ ডেটাবেস। আমরা চাইলে SQLite বাদ দিয়ে MySQL বা PostgreSQL-কে ব্যবহার করতে পারি।

SQLAlchemy-এর অফিশিয়াল ডকুমেন্টেশনের ওয়েবলিংক- <http://docs.sqlalchemy.org/en/rel_1_1/> এবং Flask-SQLAlchemy-এর অফিশিয়াল ডকুমেন্টেশনের ওয়েবলিংক- <http://flask-sqlalchemy.pocoo.org/2.1/>

### What data structures and algorithms should I know to become a good programmer?

#### Basic Category:

**Linear data structures and algorithms**

* Arrays
* Linked List
* Stack
* Queues

**Basic algorithms**

* Sorting
  + Merge Sort
  + Insertion Sort
  + Quick Sort
  + Number of inversions
* Matrix Multiplication
* Prime Sieving
* Modular Math including multiplication and division
* Euclidean Algorithm for GCD
* Modular Inverse
* Fast Exponentiation
* Fibonacci number with matrix multiplication
* Probability distribution and expected value
* Statistics
  + Mean
  + Median
  + Variance
  + Bayes theorem

**Popular Algorithmic Techniques**

* Divide and Conquer - Binary Search, Maximum Subarray
* Greedy Algorithms - Activity Selection, Huffman encoding
* Dynamic Programming - Matrix Chain Multiplication, Knapsack,
* Linear Programming - Variable Maximisation, Linear time sorting
* String Algorithms - Manacher, LCS, Edit Distance

**Non-linear Data Structures**

* Trees - Binary Tree, General Tree, Lowest Common Ancestor
* Binary Search Tree - Inorder Traversal, Level order traversal, finding kth largest element, diameter, depth, number of nodes, etc.
* Heaps - Array Implementation, Heapify, Heap Sort
* Union Find
* Hash Table - Linear Probing, Open addressing, Collision avoidance

**Graphs**

* Adjacency List, Adjacency Matrix, Weighted Edge Graphs
* Basic Traversal algorithms - Breadth First Search, Depth First Search, etc
* Shortest Path Finding Algorithm - Dijkstra, Floyd Warshal, Bellman Ford
* Minimum Spanning Tree - Kruskal's Algo, Prim's Algo

#### Advanced Category:

**Advance Tree and Graph**

* Balanced Trees - AVL, Red-Black
* Heavy Light Decomposition, B+ Trees, Quad Tree
* Advance Graph - Min Cut, Max Flow
* Maximum Matching - Hall's Marriage
* Hamiltonian Cycle
* Edge Graphs / Line Graphs
* Strongly Connected Components
* Dominant Sub-Graph, Vertex Cover, Travelling Salesman - Approx algos

**Advance String Algorithms**

* Knuth Morris Pratt Algorithm
* Rabin Karp Algorithm
* Tries and Compressed Tries
* Prefix Trees, Suffix Trees, Suffix Automation - Ukkonen Algorithm

**Advance Math**

* Fast Fourier Tranformation
* Primality Testing
* Computational Geometry - Closest point pair, Voronoi diagram, Convex Hull

**General Advance topics**

* Iterating through all combination / permutation
* Bit manipulation

# সহায়কগ্রন্থ

এই বইটির পাশাপাশি এখানে উল্লিখিত বইগুলো খুব কাজে আসবে।

১. Automate Boring Stuff with Python–দুই অংশে বিভক্ত চমৎকার একটা বই। এর এক অংশে আলোচনা করা হয়েছে ‘বেসিক পাইথন প্রোগ্রামিং’ এবং অন্য অংশে ‘অটোমেটিং টাস্কস’। অটোমেটিং টাস্কসের ভিতর ওয়েব স্ক্রাপিং, ইমেজ ম্যানিপুলেশনের মতো মজার মজার বিষয় নিয়ে আলোচনা করা হয়েছে। অনলাইনে ফ্রি পড়ুন— <https://automatetheboringstuff.com/>

২. Dive into Python 3 বইটি একেবারেই সহজপাঠ্য। মূলত পাইথন প্রোগ্রামিংয়ের মৌলিক বিষয়গুলোকে অত্যন্ত চমৎকার ও সহজভাবে তুলে ধরা হয়েছে। অনলাইনে ফ্রি পড়ুন— <http://www.diveintopython3.net/>

৩. The Hitchhiker’s Guide to Python - এটি সেই অর্থে ঠিক কোনো বই নয়, অনেকটা অনলাইন ডকের মতো, তবে এর আলোচ্যসূচি বিশাল। সম্ভবত পাইথনের কোনো দিকই এখানে বাদ দেওয়া হয়নি। পাইথন ইনস্টল করা থেকে শুরু করে, প্রজেক্ট সাজানো, কোডিং স্টাইল, কোড টেস্ট করা, নেটওয়ার্ক প্রোগ্রামিং, ওয়েব প্রোগ্রামিং, ওয়েব স্ক্রাপিং, গুই প্রোগ্রামিংসহ সবকিছুই আলোচনা করা হয়েছে। বিনা মূল্যে অনলাইনে পড়ার ওয়েবলিংক— <http://docs.python-guide.org/en/latest/>

৪. Think Python: How to Think Like a Computer Scientist – O’Reilly মিডিয়ার টেক্সটবুক ধাঁচের বই। পাইথনের মোটামুটি সব টপিক নিয়েই আলোচনা করা হয়েছে। বিনামূল্যে অনলাইনে পড়ার ওয়েবলিংক - <http://greenteapress.com/thinkpython/html/index.html>

৫. পাইথনের সবচেয়ে ভালো রিসোর্স হলো এর অফিশিয়াল ডক। পাইথনের প্রতিটি খুঁটিনাটি জিনিসের বর্ণনা অত্যন্ত চমৎকারভাবে তুলে ধরা হয়েছে। তবে বিগিনার হলে (অনেক সময় ইন্টারমিডিয়েটররাও) অফিশিয়াল ডক বুঝতে একটু কষ্ট হতে পারে। তবে বুঝতে পারলে ব্যাপক শান্তি! বিনা মূল্যে অফিশিয়াল ডক পড়ুন : <https://docs.python.org/3/> সত্যি কথা বলতে কি, ওপরের অনেকগুলো রেফারেন্সই আমি নিজে ফলো করেছি। পাইথন শেখার সময় তো করেছিই, যখন এই বইটি লিখছিলাম তখনো উল্টে-পাল্টে দেখেছি। তাদের ফলো করার চেষ্টা করেছি, অনেক ক্ষেত্রে সরাসরি তাদের দেওয়া উদাহরণই তুলে ধরেছি সগৌরবে। তাই উপরিউক্ত রেফারেন্সগুলোর গুণী লেখকদের প্রতি আমি হয়ে রইলাম চির কৃতজ্ঞ।