<https://python.maateen.me/docs/function/>

# Introduction

পাইথনে ফাংশন হলো অর্গানাইজড (organized) ও রিইউজ্যাবল (reusable) কোডের একটা ব্লক, যা নির্দিষ্ট কোনো কাজ সম্পাদন করে। ফাংশনের ভেতর আমরা হয়তো অনেক কাজই করব, কিন্তু একটির বেশি ডেটা রিটার্ন করতে পারব না বলে ফাংশনটি একটিমাত্র কাজ সম্পাদন করছে বলে ধরব। 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।

একনজরে আমরা পাইথনের দরকারি কয়েকটা বিল্ট-ফাংশন ও তাদের কাজ দেখে নেব।

| **ফাংশন** | **প্যারামিটার** | **রিটার্ন করে** |
| --- | --- | --- |
| abs() | ইন্টিজার বা ফ্লোটিং পয়েন্ট নম্বর | পরমমান |
| all() | ইটারেবল | ইটারেবল খালি হলে বা ইটারেবলের সব আইটেম True হলে True রিটার্ন করে |
| any() | ইটারেবল | ইটারেবলের যেকোনো একটি আইটেম True হলেই True রিটার্ন করবে। অন্যথায় False রিটার্ন করবে |
| enumerate() | ইটারেবল নেয়, তবে start=0 ও থাকে | ইনিউমারেট অবজেক্ট রিটার্ন করে, যেখানে ইটারেবলের প্রতিটি আইটেম ও কাউন্ট (ডিফল্টভাবে শূন্য থেকে শুরু হয়) টাপল হিসেবে থাকে |

# 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।

# Example

def print\_my\_name(myname):

# This will print the given name

print('The given name is', myname)

return

উপরের প্রোগ্রামটাকে রান করালে কিন্তু কোন আউটপুটই আমরা পাব না। এর কারণ হল আমরা ফাংশনটাকে ডিফাইন করেছি ঠিকই কিন্তু একে ডাকতে বা কল করতে ভুলে গেছি। কোন ফাংশনকে কল না করলে তার ভিতরের কোড এক্সিকিউট হয় না। এই ফাংশনটাকে কল করার জন্য ফাংশনটার নাম লিখে ব্রাকেটের ভিতর প্যারামিটারের জায়গায় ভ্যালু দিতে হবে। প্যারামিটারের জায়গায় যে ভ্যালু আমরা দেব তাই ঐ প্যারামিটারের ভ্যালু হিসাবে গণ্য হবে।

def print\_my\_name(myname):

# This will print the given name

print('The given name is', myname)

return

print\_my\_name('Maateen')

**আউটপুট**

The given name is Maateen

প্যারামিটারের জায়গায় ভ্যালু না দিয়ে আমরা চাইলে কোন ভ্যারিয়েবলও (রেফারেন্স) দিতে পারি। সেক্ষেত্রে ঐ ভ্যারিয়েবলটা আগেই ডিক্লেয়ার করে নিতে হবে।

def print\_my\_name(myname):

# This will print the given name

print('The given name is', myname)

return

name = 'Maateen'

print\_my\_name(name)

**আউটপুট**

The given name is Maateen

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।

এবার আমরা একটা ক্যালকুলেটর তৈরির কথা চিন্তা করি। আমাদের এই ক্যালকুলেটর দিয়ে তিনটা সংখ্যা যোগ করা যাবে। তো সেজন্য আমরা একটা ফাংশন লিখব। এই ফাংশন তিনটা সংখ্যা নেবে প্যারামিটার হিসাবে। আর তাদের যোগফল রিটার্ন করবে।

def add(a, b, c):

return a+b+c

temp = add(1, 2, 3)

print(temp)

**আউটপুট**

6

এই উদাহরণে আমাদের বোঝার মত অনেক কিছু আছে। তিন নাম্বার লাইনটা খেয়াল করি সবাই। add() ফাংশনটাকে কল করেছি আমরা। ফাংশনের ভিতর প্যারামিটারের ভ্যালু হিসাবে 1, 2, 3 দিয়েছি। যেহেতু প্যারামিটার পজিশনাল, তাই আমাদেরকে প্যারামিটারের ভ্যালুগুলো পজিশনাললি দিতে হবে। মানে প্যারামিটার যেটার পর যেটা আছে, ভ্যালুও সেটার পর সেটা দিতে হবে। আমাদের প্যরাামিটার ছিল a, b, c এইভাবে। তাই আমরা 1, 2, 3 এভাবে ভ্যালু পাস করায় a এর মান 1, b এর মান 2 আর c এর মান হবে 3। তারপর add() ফাংশনটা যা রিটার্ন করবে তা temp ভ্যারিয়েবলে স্টোর হবে।

# 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 add(a, b, c):

return a+b+c

temp = add(1, 2)

print(temp)

**আউটপুট**

Traceback (most recent call last):

File "/home/ugcoder/Desktop/test.py", line 3, in <module>

temp = add(1, 2)

TypeError: add() missing 1 required positional argument: 'c'

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 add(a, b, c):

return a+b+c

temp = add(b=2, c=3, a=1)

print(temp)

**আউটপুট**

6

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 add(a, b, c=3):

return a+b+c

temp = add(1, 2)

print(temp)

**আউটপুট**

6

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 add(a, b, c=3):

return a+b+c

temp = add(1, 2, 22)

print(temp)

**আউটপুট**

25

c এর মান 22 দেয়াতে নতুন একটা আউটপুট পাওয়া গেল।

### ভ্যারিয়েবল লেংথ আর্গুমেন্ট (Variable-length argument)

আমরা যে add() ফাংশনটা লিখলাম সেটাতে কিন্তু একটা ভয়ানক দোষ আছে। আমরা তিনটা মাত্র প্যারামিটার দিয়েছি। ফলে মাত্র তিনটা সংখ্যা যোগ করতে পারব। কিন্তু আমাদের ইউজার ঠিক কয়টা সংখ্যা দেবে তা কি আমরা জানি? জানিনা। তাই আমাদের উচিত এমন একটা ফাংশন লেখা যেটা যেকোন সংখ্যক আর্গুমেন্ট নিয়ে সবগুলোর যোগফল রিটার্ন করবে।

def add(\*args):

print(type(args))

tmp = 0

for number in args:

tmp = tmp + number

return tmp

temp = add(1, 2, 22, 12, 17, 21, 98)

print(temp)

**আউটপুট**

<class 'tuple'>

173

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:

def add(\*\*kwargs):

print(type(kwargs))

tmp = 0

for key in kwargs:

tmp = tmp + kwargs[key]

return tmp

temp = add(a=1, b=2, c=3, d=4)

print(temp)

**আউটপুট**

<class 'dict'>

10

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।

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)

**আউটপুট**

0

1

5

120

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।

# Some প্রবলেম সলভিং

আমরা এখন অনলাইনের অলিগলি থেকে সংগৃহীত কয়েকটা প্রবলেম ফাংশন ব্যবহার করে সলভ করার চেষ্টা করব। একটা প্রবলেম কয়েকভাবে সলভ করা যেতে পারে। এ নিয়ে রাজনীতি করার কিছু নেই!

### 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)

**আউটপুট**

Please input three integers: 12 23 34

34

Please input three integers: 67 654 9890

9890

Please input three integers: 89 564 12

564

ইউজারের কাছে থেকে আমরা স্ট্রিং হিসেবে তিনটা পূর্ণসংখ্যা ইনপুট নিয়েছি এবং a, b ও c ভ্যারিয়েবলে অ্যাসাইন করেছি। তাদের ভিতর থেকে সবচেয়ে বড় মানটি বের করতে হবে। এবার একটু ভাবা যাক। a যদি b ও c এর চেয়ে বড় হয়, তবে a-ই হবে সবচেয়ে বড় সংখ্যা। আবার b যদি a ও c এর চেয়ে বড় হয়, তবে b-ই হবে সবচেয়ে বড় সংখ্যা। অন্যথায় c অবশ্যই a ও b এর চেয়ে বড় হবে। এই লজিক ব্যবহার করেই আমরা সবচেয়ে বড় সংখ্যাটি বের করেছি।

### 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))

**আউটপুট**

Please input two integers:

20 8

4

Please input two integers:

30 40

10

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।