পাইথন ফান্ডমেন্টাল

মোহাম্মদ জাকারিয়া হোসেন

Kochu Programmer

https://youtube.com/@Kochu-Programmer

kochuprogrammer@gmail.com

https://kochu-programmer.blogspot.com

List /Array

এখন একটি ভ্যারিয়েবল তৈরি করা, তোমার জন্য কঠিন নয়। তবে তুমি ভ্যারিয়েবল এর মধ্যে একটির বেশি ডেটা কিভাবে রাখতে হবে, তা জানো না। ধরো, তুমি একটি ভ্যারিয়েবল এর মধ্যে তোমার পরিবারের সকলের বয়স রাখতে চাচ্ছো। অর্থাৎ একটি ভ্যারিয়েবল এর মধ্যে একাধিক ডেটা রাখতে চাচ্ছো। এখন তুমি কী করবে?

তোমাকে একটি লিস্ট(List) তৈরি করতে হবে। লিস্ট বা Array (অ্যারে) তৈরি করা একেবারেই সহজ। শুধুমাত্র ভ্যারিয়েবলের ভ্যালু এর জায়গায়, তৃতীয় বন্ধনীর (`[]`) মধ্যে ডেটা গুলো লিখবে। নিচে একটি উদাহরণ দেয়া হলো:

```
age = [16,23,40,18,56]
```

এইযে, আমরা একটি ভ্যারিয়েবল এরমধ্যে একাধিক ডেটা বা ভ্যালু রাখলাম, তাই এই ভ্যারিয়েবলটিকে বলা হবে List/Array/Collection. তবে আমি এটাকে লিস্ট বলবো।

একইভাবে তুমি চাইলে String এর লিস্ট তৈরি করতে পারবে। নিচে আমি তোমাদেরকে আমার পরিবারের সকলের নামের লিস্ট তৈরি করে দেখাই।

```
family = ["Zakaria", "Solaiman", "Abbu", "Ammu","Vabi","Dadi"]
print(family)
```

list কি জিনিস? সেটা তো তুমি বুঝে গেছো। এখন কমেন্টে আমাকে লিস্টের একটি real life example দিয়ে যাও।

Element

একটি লিস্টের মধ্যে থাকা, প্রত্যেকটি ডেটাকে এক একটি Element বা উপাদান বলা হয়। যেমন, `family` নামের লিস্টে `“Zakaria”` একটি Element, আবার `“Solaiman”` আরো একটি এলেমেন্ট(Element)

Index

একটি লিস্টের প্রতেকটি Element এর একটি Position(অবস্থান) রয়েছে, একে বলা হয় Index। প্রথম এলিমেন্টের ইনডেক্স হয় 0(শূন্য) এরপর এক(+1) এক(+1) করে বাড়তে থাকে। যেমন নিচের লিস্টে

```
family = ["Zakaria", "Solaiman", "Abbu", "Ammu","Vabi","Dadi"]
print(family)
```

★ “ Zakaria” এর Index হলো 0

★ “Solaiman” এর Index হলো 1

★ “Abbu” এর ইন্ডেক্স হলো 2

★ এভাবে এক এক করে বাড়বে

যদি তুমি একটি এলেমেন্টের ভ্যালু তার ইন্ডেক্স দ্বারা পেতে চাও তাহলে, প্রথমে সেই লিস্টের নাম লিখো, তারপর তৃতীয় বন্ধনীর (`[]`) মধ্যে Element টির Index নাম্বার লিখ।

```
family = ["Zakaria", "Solaiman", "Abbu", "Ammu","Vabi","Dadi"]
print(family[1])
```

output : `Solaiman`

এবার ধরুন, তোমার কাছে এমন একটি লিস্ট রয়েছে, যার Element সংখ্যা 100 টি। তাহলে গুনে গুনে ইনডেক্স বের করা কিন্তু, কষ্টসাধ্য ব্যাপার। সেজন্য ব্যবস্থা রয়েছে, প্রথমে লিস্ট(List) টির নাম লিখ, তারপর `.` (ডট) দিয়ে `index` লিখো। এরপর প্রথম বন্ধনের মধ্যে এলিমেন্টটির নাম লিখ। নিচের মতো করে :

```
family = ["Zakaria", "Solaiman", "Abbu", "Ammu","Vabi","Dadi"]
print(family.index("Ammu"))
```

এখন আমরা একটা নতুন লিস্ট নিই:

```
friends = ["Zakaria","Ibrahim","Lishan","Alhaz", "Musa"]
```

এটা আমার বন্ধুদের একটা লিস্ট, এই লিস্টের মধ্যে আমার বন্ধু লিছানের বেশ কিছু খারাপ স্বভাব আছে। তার জন্য আমরা ওকে ফ্রেন্ড সার্কেল থেকে বের করে দিচ্ছি। একই সাথে আমরা নতুন একটি ছেলেকে ফ্রেন্ড সার্কেলে যোগ করছি, ধরি তার নাম Rayed(রায়েদ)। ব্যাপারটা অনেকটি Replace করার মতো। একজন কে সরিয়ে অন্য আরেকজনকে ঢুকানো হচ্ছে। এই লিস্টের মধ্যে যদি আমরা Lishan এর জায়গায় Reyed কে Replace করতে চাই, তাহলে আমরা Index নম্বর ব্যবহার করতে পারি। নিচে মতো করে কোড লিখে:

```
friends = ["Zakaria","Ibrahim","Lishan","Alhaz", "Musa"]
friends[2] = "Rayed"
print(friends)
```

আবার ধরুন, আমাদের সাথে Bappy নামের একটি নতুন ছেলের পরিচয় হয়েছে। তাকে আমরা আমাদের ফ্রেন্ডলিস্টে যোগ করতে চাই। এর জন্য আমাদেরকে একটি বিশেষ কী-ওয়ার্ড বা ফাংশন ব্যবহার করতে হবে, যার হলো `append`

এবার চলো নিচের মতো করে Bappy কে ফ্রেন্ডলিস্টে যোগ করি।

```
friends = ["Zakaria","Ibrahim","Rayed","Alhaz", "Musa"]
friends.append("Bappy")
print(friends)
```

আবার ধরো, Musa এমন একটি কাজ করেছে, যার জন্য তাকে আমরা আমাদের ফ্রেন্ড লিস্ট থেকে বাদ দিচ্ছি। বাদ দেওয়ার জন্য যে ফাংশন টি ব্যবহার করব, সেটি হলো `remove`

আমরা Musa কে বাদ দিয়ে দিই,

```
friends = ["Zakaria","Ibrahim","Rayed","Alhaz", "Musa"]
friends.remove("Musa")
print(friends)
```

আবার কখনো কখনো আমাদের প্রয়োজন পড়বে, একটি লিস্টে কতটি উপাদান আছে সেটি বের করার। তার মানে আমরা List এর Length বের করতে যাচ্ছি। এর জন্য যে ফাংশনটি ব্যবহার করা হয় তার নাম হচ্ছে, `len` এই len হলো Length এর সংক্ষিপ্ত রূপ।

```
friends = ["Zakaria","Ibrahim","Rayed","Alhaz", "Musa","Bappy"]
count = len(friends)
print(count)
```

উপরের কোডে, আমি count নামের ভ্যারিয়েবল এর মধ্যে friends নামক লিস্টের Length রেখেছি।

List Operation

আমাদের কাছে একটি নাম্বারের লিস্ট আছে। আমরা চাই এই লিস্টের সবচেয়ে ছোট এলিমেন্ট কোনটি? সেটি জানতে। এর জন্য আমরা `min()` function ব্যবহার করব।

```
nums = [12, 54, -21, 113]
print(min(nums))
```

একইভাবে, সবচেয়ে বড় এলেমেন্টি বের করতে, `max()` ফাংশন ব্যবহার করব।

```
nums = [12, 54, -21, 113]
print(max(nums))
```

কোনো লিস্টের মধ্যে কোনো একটি উপাদান আছে কিনা সেটি যাচাই করতে আমরা `in` keyword ব্যবহার করব। প্রথমে element টি লিখবো, তারপর in,তারপর লিস্টের নাম। যদি এলেমেন্ট বা উপাদানটি লিস্টের মধ্যে থাকে, তাহলে আউটপুট আসবে, True, অন্যথায়, False

```
friends = ["Zakaria","Ibrahim","Rayed","Alhaz", "Musa","Bappy"]
print("Alhaz" in friends)
```

আবার যদি আমরা, দুইটি লিস্ট কে যোগ করতে চাই, তাহলে শুধুমাত্র `+` সাইন ব্যবহার করে করতে পারব।

নিচের কোডটি দেখলেই বুঝবে

```
first_nums = [1,2,3,4]
second_nums = [5,6,7,8]
```

```
all_nums = first_nums + second_nums
print(all_nums)
```

আবার আমরা চাইলে, লিস্টের শেষ এলিমেন্ট টিকে বের করে দিতে পারি। শুধুমাত্র `pop()` ব্যবহার করে, নিচের কোডটি দেখো।

```
nums = [ 5, 17, 18, 24, 17 ]
nums.pop()
print(nums)
```

`pop()` এর ব্রাকেটের মধ্যে যে Index নাম্বার দিবে, সেটিও মুছে যাবে,

```
nums = [ 5, 7, 18, 24, 17 ]
nums.pop(0)
print(nums)
```

উপরের কোড থেকে 5 মুছে যাবে, বিশ্বাস না করলে, কোডটি রান করিয়ে দেখ।

নিচের কোডটি ফান করিয়ে দেখতে ভুলবে না

```
nums = [ 5, 7, 18, 24, 17 ]
nums.pop(-2)
print(nums)
```

List Sort

Python-এ লিস্টকে সাজানোর জন্য কয়েকটি সহজ এবং বেসিক পদ্ধতি রয়েছে। নিচে সেগুলো ব্যাখ্যা করা হলো

1. sort() মেথড

sort() মেথড মূল লিস্টকে সরাসরি পরিবর্তন করে দেয় (in-place sorting)।

এটি ডিফল্টভাবে ছোট থেকে বড় (ascending order) সাজায়।

উদাহরণ:

```
numbers = [4, 2, 9, 1]
numbers.sort()  # লিস্ট সরাসরি সাজানো হবে
print(numbers)  # আউটপুট: [1, 2, 4, 9]
```

Descending Order: বড় থেকে ছোট সাজাতে reverse=True ব্যবহার করো।

```
numbers = [4, 2, 9, 1]
numbers.sort(reverse=True)  # লিস্ট বড় থেকে ছোট সাজানো হবে
print(numbers)  # আউটপুট: [9, 4, 2, 1]
```

1. sorted() ফাংশন

sorted() মূল লিস্ট পরিবর্তন করে না। এটি একটি নতুন লিস্ট তৈরি করে সাজানো আকারে।

উদাহরণ:

```
numbers = [4, 2, 9, 1]
sorted_numbers = sorted(numbers)  # একটি নতুন লিস্ট তৈরি হবে
print(sorted_numbers)  # আউটপুট: [1, 2, 4, 9]
print(numbers)  # আউটপুট: [4, 2, 9, 1] (মূল লিস্ট অপরিবর্তিত)
```

Descending Order: বড় থেকে ছোট সাজাতে reverse=True ব্যবহার করো।

```
numbers = [4, 2, 9, 1]
sorted_numbers = sorted(numbers, reverse=True)
print(sorted_numbers)  # আউটপুট: [9, 4, 2, 1
```

1. Custom Sorting (key ব্যবহার করে)

যদি লিস্টের আইটেমগুলোকে নির্দিষ্ট নিয়মে সাজাতে চাও, তবে key প্যারামিটার ব্যবহার করা যায়।

উদাহরণ: শব্দের দৈর্ঘ্য অনুযায়ী সাজানো:

```
names = ["John", "Alex", "Chris"]
names.sort(key=len)  # নামগুলো শব্দের দৈর্ঘ্য অনুযায়ী সাজানো হবে
print(names)  # আউটপুট: ['John', 'Alex', 'Chris']
```

Descending Order (বড় থেকে ছোট):

```
names = ["John", "Alex", "Chris"]
names.sort(key=len, reverse=True)  # বড় শব্দ আগে আসবে
print(names)  # আউটপুট: ['Chris', 'John', 'Alex']
```

Python-এ লিস্টের আইটেমগুলোকে A থেকে Z বা Alphabetical Order-এ সাজাতে খুবই সহজ। নিচে এ নিয়ে ব্যাখ্যা করা হলো:

---

1. sort() দিয়ে Alphabetical Order (A থেকে Z)

sort() মেথড ব্যবহার করে লিস্টের স্ট্রিং আইটেমগুলো সরাসরি A থেকে Z সাজানো যায়।

উদাহরণ:

```
names = ["Banana", "Apple", "Cherry", "Mango"]
names.sort()
print(names)  # আউটপুট: ['Apple', 'Banana', 'Cherry', 'Mango']
```

2. sorted() দিয়ে Alphabetical Order (A থেকে Z)

sorted() ফাংশন মূল লিস্ট পরিবর্তন না করে একটি নতুন সাজানো লিস্ট তৈরি করে।

উদাহরণ:

```
names = ["Banana", "Apple", "Cherry", "Mango"]
sorted_names = sorted(names)
print(sorted_names)  # আউটপুট: ['Apple', 'Banana', 'Cherry', 'Ma
print(names)  # আউটপুট: ['Banana', 'Apple', 'Cherry', 'Mango']
```

Case-sensitive Issue (যদি বড় অক্ষর আগে আসে)

Python-এ ডিফল্টভাবে বড় হাতের অক্ষর (uppercase) ছোট হাতের অক্ষরের (lowercase) আগে আসে।

উদাহরণ:

```
names = ["banana", "Apple", "cherry", "Mango"]
names.sort()
print(names)  # আউটপুট: ['Apple', 'Mango', 'banana', 'cherry']
```

সমাধান: Case-insensitive Sorting

key=str.lower ব্যবহার করলে বড় এবং ছোট হাতের অক্ষর একসাথে Alphabetical Order-এ সাজানো যায়।

```
names = ["banana", "Apple", "cherry", "Mango"]
names.sort(key=str.lower)
print(names)  # আউটপুট: ['Apple', 'banana', 'cherry', 'Mango']
```

List Range

range() ফাংশন দিয়ে সংখ্যা তৈরি করা যায়, যা একটি নির্দিষ্ট শুরু (start), শেষ (stop), এবং ধাপ (step) অনুযায়ী কাজ করে।

ব্যাখ্যা:

start: শুরু করার সংখ্যা (ডিফল্ট ০)।

stop: যেখানে থামবে (শেষ সংখ্যা যোগ হবে না)।

step: কত ধাপে বাড়বে বা কমবে (ডিফল্ট ১)।

উদাহরণ:

1. ০ থেকে ৯ পর্যন্ত সংখ্যা তৈরি:

```
numbers = list(range(10))
print(numbers)  # আউটপুট: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

2. ৫ থেকে ১৪ পর্যন্ত:

```
numbers = list(range(5, 15))
print(numbers)  # আউটপুট: [5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

3. ২ ধাপে:

```
numbers = list(range(0, 10, 2))
print(numbers)  # আউটপুট: [0, 2, 4, 6, 8]
```

4. উল্টো ক্রমে:

```
numbers = list(range(10, 0, -1))
print(numbers)  # আউটপুট: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

আরো কিছু List Method

extend()

আরেকটি লিস্ট বা iterable-এর উপাদানগুলো মূল লিস্টে যোগ করে।

```
my_list = [1, 2, 3]
my_list.extend([4, 5, 6])
print(my_list)
# আউটপুট: [1, 2, 3, 4, 5, 6]
```

insert()

নির্দিষ্ট ইনডেক্সে উপাদান যোগ করে।

```
my_list = [1, 2, 4]
my_list.insert(2, 3)  # 2 নম্বর ইনডেক্সে 3 যোগ হবে
print(my_list)
# আউটপুট: [1, 2, 3, 4]
```

clear()

লিস্টের সব উপাদান সরিয়ে ফেলে।

```
my_list = [1, 2, 3]
my_list.clear()
```

```
print(my_list)
# আউটপুট: []
```

count()

লিস্টে একটি নির্দিষ্ট উপাদান কয়বার আছে, তা গণনা করে।

```
my_list = [1, 2, 2, 3, 4, 2]
count = my_list.count(2)
print(count)
# আউটপুট: 3 (2 মোট ৩ বার আছে)
```

reverse()

লিস্টের উপাদানগুলোর ক্রম উল্টে দেয়।

```
my_list = [1, 2, 3, 4]
my_list.reverse()
print(my_list)
# আউটপুট: [4, 3, 2, 1]
```

copy()

লিস্টের একটি নতুন কপি তৈরি করে।

```
my_list = [1, 2, 3]
new_list = my_list.copy()
new_list.append(4)
print(my_list)
# আউটপুট: [1, 2, 3]
print(new_list)
# আউটপুট: [1, 2, 3, 4]
```

Comment

কমেন্ট হলো কোডের মধ্যে থাকা এমন কিছু লেখা যা প্রোগ্রাম রান করার সময় ইগনোর (ignore) করা হয়। এটি মূলত কোডকে ব্যাখ্যা করার জন্য বা কোডের ভেতর নোট রাখার জন্য ব্যবহার করা হয়।

কমেন্টের প্রকারভেদ

Python-এ দুই ধরনের কমেন্ট থাকে:

1. সিঙ্গেল লাইন কমেন্ট (Single-line Comment)
2. মাল্টি-লাইন কমেন্ট (Multi-line Comment)

১. সিঙ্গেল লাইন কমেন্ট
সিঙ্গেল লাইন কমেন্টে লাইনের শুরুতে একটি # চিহ্ন ব্যবহার করা হয়।
উদাহরণ:

```
# এটি একটি সিঙ্গেল লাইন কমেন্ট
print("Hello, World!")  # এটি একটি ইনলাইন কমেন্ট
```

`#` দিয়ে শুরু করা অংশ প্রোগ্রাম রান করার সময় ইগনোর হবে।

২. মাল্টি-লাইন কমেন্ট

Python-এ মাল্টি-লাইন কমেন্ট সাধারণত তিনটি ডাবল কোটেশন (""" """) বা তিনটি সিঙ্গেল কোটেশন (''' ''') ব্যবহার করে লেখা হয়।

উদাহরণ:

```
"""
এটি একটি মাল্টি-লাইন কমেন্ট।
এখানে আপনি একাধিক লাইন ব্যাখ্যা করতে পারবেন।
"""
print("Python is fun!")
```

নোট: মাল্টি-লাইন কমেন্ট প্রায়ই ডকুমেন্টেশন স্ট্রিং (docstring) হিসেবেও ব্যবহৃত হয়।

কেন কমেন্ট ব্যবহার করব?

1. কোডের ব্যাখ্যা প্রদান:

আপনার কোড কী করছে তা ব্যাখ্যা করতে।

ভবিষ্যতে অন্য প্রোগ্রামার বা নিজের জন্য কোড বোঝা সহজ হয়।

2. ডিবাগিং সহজ করা:

কোডের নির্দিষ্ট অংশ সাময়িকভাবে ইগনোর করার জন্য।

For Loop

ধরো তোমাকে তোমার প্রাইভেট টিউটর ফোন করে বলেছেন,"সে আজকে পড়াতে আসবে না, তুমি যেন সবাইকে ফোন করে জানিয়ে দাও " । এর জন্য তুমি একের পর একজনকে ফোন করছো খেয়াল করে দেখ তুমি বার বার একই কাজ করছো। অর্থাৎ তুমি একটি লুপের মধ্যে আছো।
যখন কাজ বার বার করা হয় তখন, তাকে লুপ বলে।।

একটু স্কুল কলেজের মাস্টার মশাইদের স্টাইলে বললে,

লুপ (Loop) প্রোগ্রামিংয়ের একটি গুরুত্বপূর্ণ ধারণা, যা একটি নির্দিষ্ট কাজ বা নির্দেশাবলী বারবার পুনরাবৃত্তি করতে ব্যবহৃত হয়। সাধারণত, যখন কোনো কাজ বারবার করা প্রয়োজন হয়, তখন সেই কাজটি বারবার ম্যানুয়ালি না লিখে লুপ ব্যবহার করা হয়। লুপ মূলত কোডকে সংক্ষিপ্ত, কার্যকর এবং পড়তে সহজ করে তোলে।

লুপের ধরন
প্রোগ্রামিং ভাষাগুলিতে সাধারণত তিন প্রকার লুপ দেখা যায়:
1. For Loop:
2. While Loop
3. Do-While Loop(পাইথনে নেই)

এখন চলো কিছুটা কোডিং করে বুঝে নিই ফোর লুপ কীভাবে কাজ করে.

```
contact_list = ["Ahad","Bappy","Afrin","Ibrahim"]
for call in contact_list:
    print(call)
```

For loop Structure

এটাকে ফোর লুপ বলা হয়, কারণ এটি `for` কী-ওয়ার্ড দিয়ে শুরু হয়।

একটি for loop লিখতে তোমাকে পাঁচটি কাজ করতে হবে:

1. `for` কীওয়ার্ডটি লিখতে হবে।
2. একটি লুপ ভ্যারিয়েবল তৈরি করতে হবে, এই ভ্যারিয়েবল এর মান হবে লিস্টে থাকা প্রত্যেকটি উপাদান।
3. `in` কীওয়ার্ডটি লিখতে হবে।
4. যে লিস্টটিতে for লুপ রান করাতে চাও, সেই লিস্টের নাম লিখতে হবে। এরপর একটি কোলন (:) দিতে হবে।
5. Tab প্রেস করে indent করতে হবে, এরপর যে কাজটি বারবার করতে চাও, সেটির জন্য কোডিং করতে হবে।

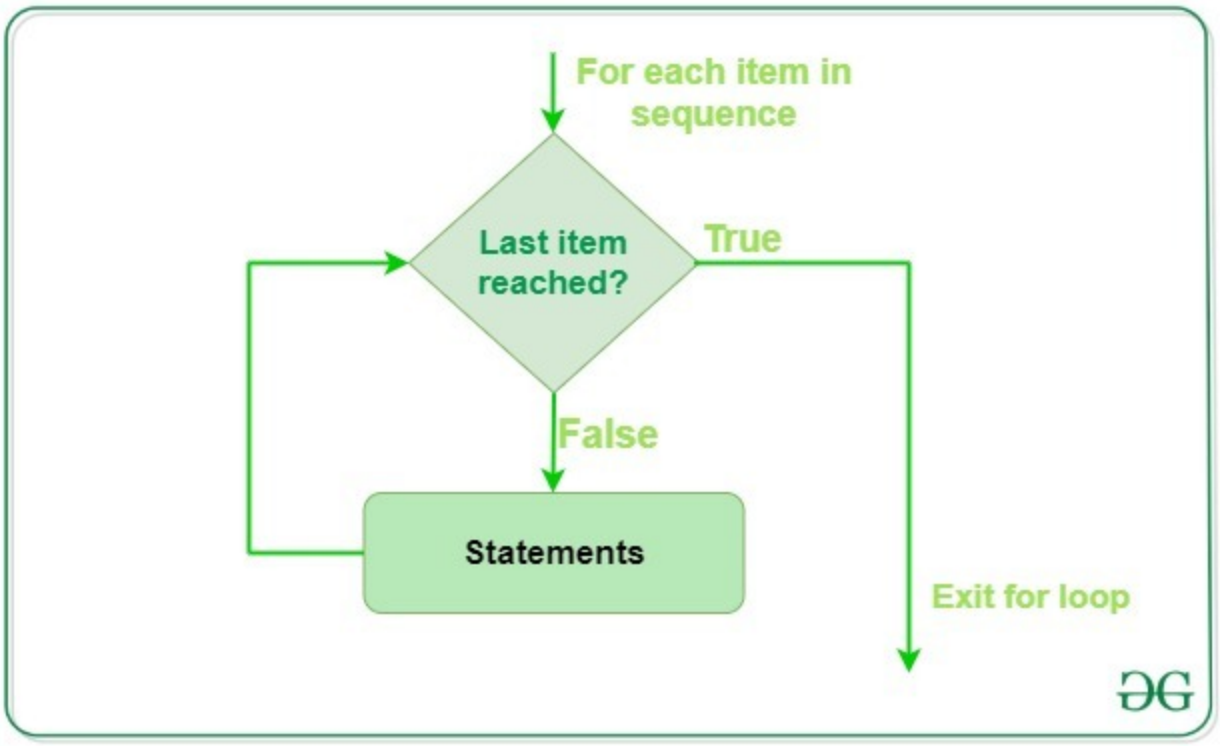
আরো কিছু কোডিং উদাহরণ:

```
# iterate from i = 0 to i = 3
for i in range(4):
    print(i)
```

```
language = 'Python'

# iterate over each character in language
for x in language:
    print(x)
```

নিজের ফলো চার্টটি দেখে বুঝে নাও, for লুপ কিভাবে কাজ করে?



https://www.geeksforgeeks.org এর সৌজন্যে

মারো ব্রেক Break

যেহেতু আমরা for loop এর মধ্যে অনেক কিছুই লিখতে পারি, তাই অনায়াসে একটি কন্ডিশনাল স্টেটমেন্টও দিয়ে দেওয়া যায়। নিচের কোডটি দেখ, যখন কোনো একটি সংখ্যা শূন্যের চেয়ে বড় হবে, তখন-ই মাত্র কোডটি print করবে।

```
nums = [2, 74, -21, 213]
for num in nums:
   if num > 0:
     print(num)
```

উপরের কোডে, for loop. এর মধ্যে থাকা Conditional Statement এ চেক করা হচ্ছে যে, num, শূন্য(০) থেকে বড় কিনা। যদি বড় হয়, তাহলে num কে প্রিন্ট করো। আর ছোট হলে প্রিন্ট করা লাগবে না।

যখন এই লুপটি শুরু তখন, লুপ ভ্যারিয়েবলের(num) মান হিসেবে পাই 2 কে, 2 যেহেতু 0 থেকে বড় তাই এটি প্রিন্ট হবে। এখই ঘটনা ঘটবে 74 এর সাথে।

যখন লুপটি -21 এ পৌঁছাবে, তখন এটি প্রিন্ট হবে না। কারণ -21, শূন্য থেকে ছোট। আবার 213 যেহেতু শূন্য থেকে বড় তাই এটি প্রিন্ট হবে।

লুপ শেষে আমরা আউটপুট পাবো

```
2
74
213
```

এখন আমরা চাই, যখন-ই আমাদের for loop কোনো ঋণাত্মক নাম্বার(negative number) পাবে, তখনই যেন থেমে যায়। আমরা কীভাবে থামাতে পারি? এটিই হলো তোমার প্রশ্ন। দেখো ভাইয়া/আপু প্রত্যেকটি গাড়িকে থামানোর জন্য একটি ব্রেক(break থাকে। আর for loop কে থামানোর জন্যও আছে `break` কী-ওয়ার্ড। নিচের কোডটি PyCharm কিংবা যেকোনো একটি কোড এডিটরে রান করালে বুঝবে।

```
nums = [2,  74, -21, 213]
for num in nums:
   if num < 0:
     break
   print(num)
```

তোমাকে আরো একটি উদাহরণ দিই:

ধরা যাক, আমরা ১ থেকে ১০ পর্যন্ত সংখ্যা প্রিন্ট করব, তবে যখন সংখ্যা ৫ এ পৌঁছাবে, তখন লুপটি থেমে যাবে।

```
for i in range(1, 11):  # 1 থেকে 10 পর্যন্ত সংখ্যা
    if i == 5:          # যদি i == 5 হয়
```

```
        break             # লুপ থামিয়ে দাও
    print(i)             # সংখ্যাটি প্রিন্ট করো
```

আউটপুট

```
1
2
3
4
```

কীভাবে কাজ করে:
1. for লুপ 1 থেকে 10 পর্যন্ত সংখ্যা ইটারেট করে।

2. if i == 5 শর্তটি সত্য হলে, break কার্যকর হয়।

3. break লুপ বন্ধ করে এবং পরবর্তী কোডে চলে যায়।

বাস্তব ব্যবহার:
break সাধারণত এমন ক্ষেত্রে ব্যবহৃত হয় যেখানে নির্দিষ্ট শর্তে লুপ থামানো প্রয়োজন। যেমন, সার্চ অপারেশন, ইনফিনিট লুপ(একটু পরেই জানব) থেকে বের হওয়া ইত্যাদি।

এড়িয়ে যাই চলো (continue)

আমাদের আশেপাশে এমন কিছু লোক থাকে যাদেরকে আমরা এড়িয়ে যেতে চাই। কখনো তাদের-কে না দেখার ভান করে, কখনো বা ব্যস্ততা দেখিয়ে। যাইহোক, আমরা এখন চাই যে, আমাদের for loop যখন-ই কোনো ঋণাত্মক সংখ্যা পাবে, তখন তাদেরকে যেন এড়িয়ে যায়। এই এড়িয়ে যাওয়ার জন্য ব্যবহৃত কী-ওয়ার্ডটি হলো `continue`

নিচের কোডটি দেখ,

```
nums = [1, 22, -3, -21, -45, 35, -18]
for num in nums:
   if num < 0:
```

```
        continue
    print(num)
```

কি ভাইয়া/আপু বুঝতে অসুবিধা হচ্ছে? হলে নিচের লিখা টুকু পড়ো:

নিচের প্রোগ্রামে continue কীভাবে কাজ করছে এবং আউটপুট কী হবে তা ধাপে ধাপে দেখানো হলো।

তোমার কোড:

```
nums = [1, 22, 35, -21, 45, 16, 18]
for num in nums:
    if num % 2 == 0:
        continue
    print(num)
```

কোডের কাজ:

1. nums লিস্টে কিছু সংখ্যা আছে: [1, 22, 35, -21, 45, 16, 18]।
2. for লুপটি লিস্টের প্রতিটি সংখ্যার উপর ইটারেট করছে।
3. if num % 2 == 0 শর্তটি চেক করছে, সংখ্যা জোড় (even) কিনা।

যদি জোড় সংখ্যা হয়, তাহলে continue কার্যকর হবে এবং লুপ পরবর্তী সংখ্যায় চলে যাবে।

যদি বিজোড় (odd) সংখ্যা হয়, তাহলে print(num) কার্যকর হবে এবং সংখ্যাটি প্রিন্ট হবে।

ধাপে ধাপে ব্যাখ্যা:

1. num = 1:

1 % 2 == 0 → শর্ত মিথ্যা।

print(1) → আউটপুট: 1।

2. num = 22:

22 % 2 == 0 → শর্ত সত্য।

continue কার্যকর → লুপ পরবর্তী সংখ্যায় চলে যাবে, কিছু প্রিন্ট হবে না।

3. num = 35:

35 % 2 == 0 → শর্ত মিথ্যা।

print(35) → আউটপুট: 35।

4. num = -21:

- 21 % 2 == 0 → শর্ত মিথ্যা।

print(-21) → আউটপুট: -21।

5. num = 45:

45 % 2 == 0 → শর্ত মিথ্যা।

print(45) → আউটপুট: 45।

6. num = 16:

16 % 2 == 0 → শর্ত সত্য।

continue কার্যকর → কিছু প্রিন্ট হবে না।

7. num = 18:

18 % 2 == 0 → শর্ত সত্য।

continue কার্যকর → কিছু প্রিন্ট হবে না।

আউটপুট আসবে

```
1
35
-21
45
```

এখনো যদি বুঝতে অসুবিধা হয় তাহলে, শুধু এটুকু মনে রেখো:

`continue` লুপের বর্তমান ইটারেশন স্কিপ করে পরবর্তী ইটারেশনে চলে যায়।
বিজোড় সংখ্যাগুলো প্রিন্ট হয়েছে, কারণ num % 2 == 0 শর্ত তাদের জন্য মিথ্যা ছিল।
জোড় সংখ্যাগুলো স্কিপ করা হয়েছে।

While loop

ধরো, তুমি সকাল দশটায় ইউটিউবে ঢুকলে ১ ঘন্টার রিমাইন্ডার সেট করে। যে তুমি একটার পর একটা ভিডিও দেখবে ১১ টা পর্যন্ত। এখানে তুমি একটার পর একটা ভিডিও দেখছো, তার মানে তুমি একটি লুপোর(loop) মধ্যে আছো। তবে, তোমার লুপে একটি শর্ত আছে, তা হলো লুপটি চলবে ১ ঘন্টা। এর মানে হচ্ছে, তুমি **While Loop** এ আছো।

যখন কোনো একটি কাজ একটি শর্তের ভিত্তিতে বারবার করা হয়, তখন তাকে While Loop বলা হয়।

অন্যভাবে বললে, while লুপ হলো প্রোগ্রামিংয়ের একটি গুরুত্বপূর্ণ অংশ যা একটি শর্ত (condition) সত্য হওয়া পর্যন্ত নির্দিষ্ট কোড বারবার চালাতে সাহায্য করে। Python-এ এটি সাধারণত এমন কাজ করতে ব্যবহৃত হয় যেখানে আমরা জানি না লুপ কতবার চলবে, তবে আমরা একটি শর্ত দিয়েই এটি নিয়ন্ত্রণ করি।

নিচে while loop এর একটি উদাহরণ দেওয়া হলো :

```
count = 0
while count < 5:
  print (count)
  count = count + 1
```

While Loop Structure

While loop লিখতে তোমাকে ৫ টি কাজ করতে হবে :

1. প্রথমে একটি loop variable তৈরি করতে হবে।
2. while কী-ওয়ার্ডটি লিখতে হবে।
3. while এর পরে থামার জন্য একটি কন্ডিশন লিখতে হবে বা শর্ত লিখতে হবে।
4. while এর নিচে, যে কাজটি বারবার করতে হবে, সেটি লিখতে হবে।
5. loop variable কে আপডেট করতে হবে। অন্যথায়, loop কোনো দিনও শেষ হবে না।

চলো, নিচের প্রোগ্রামটির চুল-ছেড়া বিশ্লেষণ করি:

```
count = 0
while count < 5:
  print (count)
  count = count + 1
```

ব্যাখ্যা:

1. count = 0
   এখানে count নামে একটি ভেরিয়েবল তৈরি করা হয়েছে এবং এর মান 0 দেওয়া হয়েছে।
2. while count < 5
   লুপটি চালু হবে যতক্ষণ পর্যন্ত count ভেরিয়েবলের মান 5-এর চেয়ে ছোট থাকে।
3. লুপের ভিতরের কাজ:

print(count)
প্রতিবার লুপের ভিতরে count ভেরিয়েবলের মান প্রিন্ট করবে।

count = count + 1
প্রতিবার লুপ চলার পরে count-এর মান ১ করে বাড়বে।

4. লুপ বন্ধ হওয়া:
   যখন count ভেরিয়েবলের মান 5 বা তার চেয়ে বড় হবে, তখন while লুপটি বন্ধ হয়ে যাবে।

লুপের আউটপুট:

প্রতিটি স্টেপে count ভেরিয়েবলের মান হবে:

প্রথমবার: 0

দ্বিতীয়বার: 1

তৃতীয়বার: 2

চতুর্থবার: 3

পঞ্চমবার: 4

তখন লুপ বন্ধ হবে, কারণ count = 5 হয়ে গেছে, যা শর্ত count < 5 পূরণ করে না।

```
0
1
2
3
4
```

While loop অল্প কিছু ব্যবহার

তুমি চাইলেই while loop ব্যবহার করে, ১ থেকে ১০ পর্যন্ত সব সংখ্যার যোগফল বের করতে পারো।

```
i = 1
sum = 0
while i <= 10:
   sum = sum + i
   i = i + 1
print (sum)
```

এক থেকে দশ পর্যন্ত, সকল জোড় সংখ্যার যোগফল বের করতে চাইলে

```
i = 1
sum = 0
while i <= 10:
   if i%2 == 0:
      sum = sum + i
   i = i+1

print(sum)
```

১ থেকে ১০ পর্যন্ত সকল বিজোড় সংখ্যার যোগফল বের করতে চাইলে,

```
i = 1
sum = 0
while i <= 10:
    if i%2 != 0:
        sum = sum + i
    i = i+1

print(sum)
```

While loop এ `break` ব্যবহার করতে পারবে,

```
count = 0

while count <15:
   if count == 5:
      break
   print(count)
   count += 1
```

`Continue` ব্যবহার করতে চাইলে,

```
count = 0
while count <= 10:
   count = count + 1
   if count % 2 == 0:
      continue
   print(count)
```

এমন-কি-হতে পারে না যে, তুমি যাও লুপটি আজীবন চলুক, লুপটির যেন কোনো শেষ না হয়। এই ধরণের লুপকে বলে, infinity loop.

Infinity loop তৈরি করাও সহজ, while এরপর কোনো কন্ডিশন দিবেনা, বরং লিখবে `True`, এইতো কাজ শেষ।

```
while True:
  print("I love অমুক-তমুক")
```

While ও for লুপ কোনটি কখন ব্যবহার করব?

তুমি কখন For Loop এবং কখন While Loop ব্যবহার করবে, তা নির্ভর করে তোমার সমস্যার ধরন এবং কীভাবে তুমি লুপটি চালাতে চাও তার ওপর। নিচে ব্যাখ্যা করা হলো:

For Loop

যখন তুমি জানো কতবার লুপটি চালাতে হবে বা কোনো নির্দিষ্ট সিকোয়েন্স (যেমন লিস্ট, টাপল, রেঞ্জ) নিয়ে কাজ করতে চাও, তখন For Loop ব্যবহার করা উত্তম।

উদাহরণ:

```
# ১ থেকে ১০ পর্যন্ত সংখ্যাগুলোর যোগফল বের করা
sum = 0
for i in range(1, 11):
    sum += i
print(sum)  # আউটপুট: ৫৫
```

এখানে তুমি জানো লুপটি ১০ বার চলবে। এজন্য For Loop সবচেয়ে উপযুক্ত।

While Loop

যখন তুমি জানো না লুপটি কতবার চলবে, বরং শর্ত পূরণ হওয়া পর্যন্ত চালাতে চাও, তখন While Loop ব্যবহার করবে।

উদাহরণ:

```
# যতক্ষণ পর্যন্ত একটি সংখ্যা ১০-এর কম থাকে, তা দ্বিগুণ করো
num = 1
while num < 10:
    num *= 2
    print(num)
```

এখানে তুমি নিশ্চিত নও লুপটি কতবার চলবে, কারণ এটি শর্তের ওপর নির্ভর করছে।

তুলনা ও সিদ্ধান্ত

1. For Loop:

যখন নির্দিষ্ট রেঞ্জ বা সিকোয়েন্স নিয়ে কাজ করছ।

উদাহরণ: লিস্ট, স্ট্রিং, রেঞ্জের প্রতিটি আইটেমের ওপর কাজ করা।

1. While Loop:

যখন একটি শর্ত পূরণ হওয়া পর্যন্ত লুপ চালাতে হবে।

উদাহরণ: ইনফিনিট লুপ বা ইউজার ইনপুটের ওপর নির্ভরশীল কাজ।

Function

ফাংশন হলো কোডের একটি নির্দিষ্ট অংশ, যা নির্দিষ্ট কাজ সম্পাদনের জন্য লেখা হয়। আপনি একবার ফাংশন তৈরি করলে এটি বারবার ব্যবহার করতে পারেন। এটি কোডকে সংগঠিত ও পুনঃব্যবহারযোগ্য করে তোলে।

উদাহরণ দিয়ে বলি, তুমি প্রতিদিন স্কুলে যাও। স্কুলে যাওয়ার জন্য, তোমাকে তোমার মা একটা নির্দেশ দেন যে, “Prepare for School”(স্কুলে যাওয়া প্রস্তুতি নাও)।

তাহলে তোমার Task হচ্ছে prepare for school

কিন্তু স্কুলে যাওয়ার প্রস্তুতির জন্য তুমি নিচের কাজগুলোও করো:

1. ঘুম থেকে উঠো
2. দাঁত ব্রাশ কর
3. গোসল করো
4. স্কুল ইউনিফর্ম পড়ো
5. সকালের খাবার খাও
6. ব্যাগ গুছিয়ে নাও
7. সবশেষে রওনা হও

উপরের এই কাজগুলো হচ্ছে Sub-tasks. তার মানে main task (Prepare For School) এর অনেক গুলো sub-task আছে।

কিন্তু কমান্ড লাইন হলো Prepare For School. এটিই হলো ফাংশন।

কীভাবে Function লিখবে?

প্রথমে `def` কী-ওয়ার্ডটি লিখবে, তারপর function টির। একটি নাম দিবে। এরপর `()` দিয়ে কোলন(:)

এই লাইনের নিচে কোড ইন্ডেন্ট করে Sub-task গুলো লিখবে। ফাংশনটি কল করতে চাইলে Function এর নাম লিখার পর বন্ধনী `()` দিবে।

নিচের মতো করে:

```
def prepare_for_school():
    print("wake up")
    print("Brush Teeth")
    print("Take a Shower")
    print("eat breakfast")
    print("leave for school")

#call the function
prepare_for_school()
```

Parameter

প্যারামিটার হলো সেই তথ্য যা আমরা ফাংশনে পাঠাই, যাতে ফাংশন কাজ করতে পারে।

সহজ উদাহরণ:

ধরো, আমরা এমন একটি ফাংশন তৈরি করব, যেটা নাম নিয়ে সেই নামকে প্রিন্ট করবে।

```
def greet(name):  # এখানে 'name' হলো প্যারামিটার
    print(f"Hello, {name}!")
greet("Rahim")

#আউটপুট: Hello, Rahim!
```

---

return

তুমি একটি রেস্টুরেন্টে গিয়ে ইনপুট হিসেবে টাকা দিলে, রেস্টুরেন্ট তোমাকে মজার মজার খাবার দিবে।

একটি মিক্সার বা বিলিন্ডারে কমলা লেবু আর বরফ দিয়ে ঘুললি দিলে, বিলিন্ডার থেকে পাবে ঠান্ডা লেবুর শরবত। খেয়াল করো, রেস্টুরেন্ট এবং বিলিন্ডার উভয়েই তোমাকে কিছু return দিচ্ছে।

function থেকেও চাইলে return নেওয়া যায়, নিচের মতো করে:

```
def add(a, b):
   return a + b

sum = add(33, 66)
print(sum)
```

এই কোডটি একটি সাধারণ Python প্রোগ্রাম, যা দুটি সংখ্যা যোগ করে এবং তার ফলাফল প্রিন্ট করে। প্রতিটি অংশের ব্যাখ্যা নিচে দেওয়া হলো:

1. ফাংশন ডিফাইন করা:

```
def add(a, b):
   return a + b
```

`def`: এটি একটি ফাংশন তৈরি করার জন্য ব্যবহৃত কীওয়ার্ড।

`add(a, b)`: এখানে add হল ফাংশনের নাম, এবং a এবং b হল প্যারামিটার, যেগুলো ফাংশনের ভিতরে মান পাস করার জন্য ব্যবহৃত হয়।

return a + b: এটি a এবং b যোগ করে ফলাফল ফিরিয়ে দেয়।

2. ফাংশন কল করা এবং মান সংরক্ষণ করা:

sum = add(33, 66)

এখানে add(33, 66) ফাংশনটি কল করা হয়েছে এবং a=33 এবং b=66 পাস করা হয়েছে।

ফাংশনটি 33 + 66 যোগ করে 99 রিটার্ন করে।

এই রিটার্ন করা মানটি sum নামক ভেরিয়েবলে সংরক্ষণ করা হয়।

3. প্রিন্ট করা:

print(sum)

এখানে print(sum) কমান্ডটি sum ভেরিয়েবলের মান প্রিন্ট করে, যা 99।

আউটপুট:

99

এই প্রোগ্রামের মাধ্যমে দুটি সংখ্যা যোগ করার জন্য একটি ফাংশন ব্যবহার দেখানো হয়েছে।

বিজোড় সংখ্যা কিনা, সেটি চেক করার ফাংশন

```
def is_odd(num):
    if num % 2 == 0:
        return False
    else:
        return True

# Example usage
print(is_odd(7))  # This will return True because 7 is an odd nu
```

দেখতো নিজে বুঝতে পারো কিনা:

```
def add (a,b):
    return a + b

x = add(2,3)
y = add(4,5)
z = add(x, y)
print(z)
```

কেন return

return ফাংশনের আউটপুটকে বাইরে ফিরিয়ে আনে, যাতে তা সংরক্ষণ বা পুনরায় ব্যবহার করা যায়।
print শুধু স্ক্রিনে আউটপুট দেখায়, পরবর্তী প্রসেসিংয়ের জন্য তা ব্যবহৃত হয় না।

একের মধ্যে আরেক

তুমি চাইলে একটি ফাংশনের মধ্যে অন্য একটি ফাংশনকে কল করতে পারো।

```
def is_odd(num):
    if num % 2 == 0:
        return False
    else:
        return True


def evenify(num):
    check_odd = is_odd(num)
    if check_odd == True:
        even_num = num * 2
    else:
        even_num = num
    return even_num


result = evenify( 3)
print(result)
```

কোডটি একটি সংখ্যাকে "জোড়" (even) করতে ব্যবহৃত হয়। কোডের কাজ নিম্নরূপ:

1. is_odd(num) ফাংশন:

একটি সংখ্যা বিজোড় (odd) কিনা তা যাচাই করে।

যদি সংখ্যা জোড় হয়, এটি False রিটার্ন করে।

যদি সংখ্যা বিজোড় হয়, এটি True রিটার্ন করে।

2. evenify(num) ফাংশন:

প্রথমে is_odd(num) ফাংশনের মাধ্যমে যাচাই করে সংখ্যা বিজোড় কিনা।

যদি বিজোড় হয়, সংখ্যাটিকে ২ দিয়ে গুণ করে (বিজোড় সংখ্যাকে জোড় করে)।

যদি জোড় হয়, সংখ্যাটি অপরিবর্তিত থাকে।

পরিবর্তিত বা অপরিবর্তিত সংখ্যাটি রিটার্ন করে।

3. মূল প্রোগ্রাম:

evenify(3) কল করে।

৩ একটি বিজোড় সংখ্যা, তাই এটি ২ দিয়ে গুণ করে ৬ রিটার্ন করে।

print(result) আউটপুট দেখায়: 6