

Class #06: Functions in Python

1. Defining functions in Python.
2. Using function arguments.
3. Understanding recursion in Python.
4. Exploring built-in functions.
5. Anonymous functions or lambda expressions. Assignment #6:

Python-এ iterable হলো এমন একটি অবজেক্ট, যা এক এক করে উপাদান প্রদান করতে পারে। সহজভাবে বলতে গেলে, iterable হল এমন একটি বস্তু যেটি লুপের(for/while) মাধ্যমে বা অন্য কোনো পদ্ধতিতে তার উপাদানগুলো একে একে অ্যাক্সেস করা যেতে পারে।

যেসব ডেটা স্ট্রাকচার iterable হতে পারে, তার মধ্যে কিছু সাধারণ উদাহরণ হল:

- List (তালিকা)
- Tuple (টাপল)
- String (স্ট্রিং)
- Set (সেট)
- Dictionary (ডিকশনারি)

Iterable এর কিছু বৈশিষ্ট্য:

1. একটি একটি উপাদান অ্যাক্সেস করা যায়: iterable এর উপাদানগুলো এক এক করে ট্রাভার্স করা (অথবা পড়া) সম্ভব।
2. iter() এবং next() ব্যবহার করা যায়: একটি iterable এর উপাদানগুলো পরপর নিতে iter() ও next() ফাংশন ব্যবহার করা যেতে পারে।
3. ইটেরেবল ডেটা স্ট্রাকচারের মধ্যে উপাদান থাকতে হবে: যেমন তালিকা বা স্ট্রিং এর মধ্যে আলাদা আলাদা মান থাকে, যেগুলো একে একে বের করা যায়।

সাধারণভাবে বলতে গেলে, যদি কোনো ডেটা স্ট্রাকচার একে একে তার উপাদানগুলো দিতে পারে, তবে সেটি iterable।

1. Defining functions in Python.

Python-এ built-in functions হলো এমন functions যা Python ভাষায় স্বাভাবিকভাবেই অন্তর্ভুক্ত থাকে এবং আপনাকে এগুলি ব্যবহার করতে কোনো আলাদা লাইব্রেরি বা মডিউল import করতে হয় না। এই functions আপনাকে বিভিন্ন সাধারণ কাজ সহজে সম্পন্ন করতে সাহায্য করে।

সোজা বাংলায় কোনো শব্দের পরে parenthesis/প্রথম বন্ধনী () থাকলে সেসব শব্দকে তাকে function বলে।

print(), input(), len(), type(), str(), list(), tuple(), set(), dict() এদের সবার পরেই parenthesis বা প্রথম বন্ধনী আছে। সেজন্য, তারা সবাই function.

For more details: <https://www.geeksforgeeks.org/python-functions/?ref=roadmap>

How to create a function in python:

পাইথনে ফাংশন তৈরি করার জন্য def কিওয়ার্ড ব্যবহার করা হয়। একটি ফাংশন তৈরির জন্য প্রথমে def লিখে, এরপর ফাংশনের নাম এবং প্যারামিটার (যদি থাকে) তারপর একটি কলন (:) ব্যবহার করে ফাংশনের কটেন্ট লেখা হয়। ফাংশনের কাজ সাধারণত ইনডেন্টেশন (indentation) দিয়ে নির্দেশিত হয়। এর মানে হলো, ফাংশনের কাজ তার পেটের ভিতরের যা লিখা আছে, তা দিয়ে নির্দেশিত হয় নিচে একটি সাধারণ ফাংশন তৈরি করার উদাহরণ দেওয়া হল:

সাধারণ ফাংশন উদাহরণ:

```
def minhaz():
    print("হ্যালো, শুভ সকাল!")
```

এখানে:

- ✓ def কিওয়ার্ড দ্বারা ফাংশন শুরু হয়েছে। define এর বাংলা অর্থ তৈরি করা।
- ✓ minhaz() হচ্ছে ফাংশনের নাম। আমরা যেকোনো নাম দিতে পারি।
- ✓ কোন প্যারামিটার নেই, তাই খালি বন্ধনী () ব্যবহৃত হয়েছে।
- ✓ ফাংশনের পেটের মধ্যে (indentation) print("হ্যালো, শুভ সকাল!") কমান্ডটি ফাংশনের কাজ।

function বানানো ও function call করা:

যেমন আপনি মনে মনে ঠিক করেন, কাউকে শুভ সকাল জানাবেন। শুধুমাত্র এই পরিকল্পনা করলেই তা বাস্তবায়িত হয় না, আপনাকে সেই শুভ সকালটি সঠিকভাবে জানাতে হবে। ঠিক তেমনি, প্রোগ্রামিংয়ে একটি function তৈরি (define) করা হয় যখন আমরা সেই function-এর কাজ বা কার্যকলাপ নির্ধারণ করি। কিন্তু, functionটি শুধুমাত্র তৈরি করলে কাজ হবে না, সেটিকে কার্যকর করার জন্য আমরা ফাংশন কল (function call) করতে হয়।

যখন আপনি একটি ফাংশনের নাম লিখে সেটি কল (call) করেন, তখন সেই ফাংশনটি কার্যকরী হয় এবং নির্ধারিত কাজটি সম্পন্ন হয়। যেমন, আপনি শুভ সকাল বলার সিদ্ধান্ত নিলেন, এখন সেটি বলার জন্য আপনাকে বাস্তবভাবে উচ্চারণ করতে হবে। একে প্রোগ্রামিং ভাষায় "ফাংশন কল" বলে।

সংক্ষেপে:

- ✓ ফাংশন তৈরি (Define): ফাংশনটি কী কাজ করবে তা ঠিক করা।
- ✓ ফাংশন কল (Call): ফাংশনটির কাজ বাস্তবায়ন করা বা কার্যকরী করা।

এভাবে ফাংশন কল করা একটি ফাংশনকে কার্যকর করে তোলে, যা প্রোগ্রামিংয়ের একটি গুরুত্বপূর্ণ অংশ।

```
#define minhaz function
def minhaz():
    print("হালো, শুভ সকাল!")
#calling minhaz function
minhaz()
```

```
def minhazul():
    n = int(input("enter a number: "))
    for x in range(n):
        print(x, end=" ")
```

এটা চলবে না, কারণ call দেয়া হয় নাই।

নিচের প্রোগ্রাম এ input এ তুমি 50 দিবা। এবং শুধুমাত্র output আমাকে whats app এ মেসেজ করে দিবা।
এটা তোমার কাজ

```
def minhazul():
    n = int(input("enter a number: "))
    for x in range(n):
        print(x, end=" ")
minhazul()
```

এটা চলবে, কারণ call দেয়া হয়েছে।

2. Using function arguments.

ফাংশন একটি প্রোগ্রামিং কনসেপ্ট, যা এক বা একাধিক ইনপুট নিয়ে একটি নির্দিষ্ট কাজ বা প্রক্রিয়া সম্পাদন করে এবং একটি আউটপুট প্রদান করে। এটি প্রোগ্রামিং ভাষায় কোডের একটি পুনর্ব্যবহারযোগ্য ব্লক, যা বিভিন্ন জায়গায় ব্যবহার করা যেতে পারে।

Argument (আর্গুমেন্ট) কী?

Argument হলো সেই মান বা তথ্য যা একটি ফাংশন বা মেথডে পাস করা হয় যখন ফাংশনটি কল করা হয়। এটি ফাংশনের first bracket() ভিতরে ব্যবহৃত হয় এবং ফাংশনটির কার্যকারিতা নির্দিষ্ট করতে সাহায্য করে।

উদাহরণস্বরূপ, **print()** এবং **input()** ফাংশনে যে মান বা তথ্য আমরা () ব্রাকেটের মধ্যে দেই, তা হলো আর্গুমেন্ট। () ব্রাকেটের মধ্যে যা লিখবো তা হচ্ছে, আর্গুমেন্ট।

Argument (আর্গুমেন্ট) এর বাস্তব উদাহরণ: ধরা যাক, আপনি একটি বইয়ের দোকানে গিয়ে একজন দোকানদারকে বললেন, "দয়া করে আমাকে একটি সায়েন্স বই দিন।"

এখানে, **বইয়ের ধরন** (যেমন: সায়েন্স, রূপকথা, ইতিহাস) হলো আপনার argument, যেটি দোকানদারকে বলে দিচ্ছে আপনি কোন ধরনের বই চান। দোকানদার সেই অনুযায়ী নির্দিষ্ট বইটি আপনার হাতে তুলে দেবে।

এভাবে, argument হলো সেই তথ্য বা মান যা ফাংশনে পাস করা হয়, যাতে ফাংশনটি নির্দিষ্টভাবে কাজ করতে পারে।

বাস্তব উদাহরণ:

ধরা যাক, আপনি একটি বিয়ের অনুষ্ঠানে গিয়ে বিয়ে করার প্রক্রিয়া দেখতে চান। এখানে, বিয়ে হতে পারে একটি ফাংশন। ফাংশনের ইনপুটগুলো হবে—

1. বর (Groom)
2. কনে (Bride)
3. তারিখ (Date)
4. স্থান (Venue)

এবং ফাংশনের আউটপুট হবে—

✓ বিয়ের অনুষ্ঠান সম্পন্ন (Wedding Ceremony Completed)

In programming (define function):

```
# Define a function to print the wedding details
def wedding(groom, bride, date, venue):
    # Printing the wedding details with a formatted string
    print(f"Brude: {bride} and Groom: {groom} will marry on {date} at {venue}. Wedding
Ceremony Completed!")
```

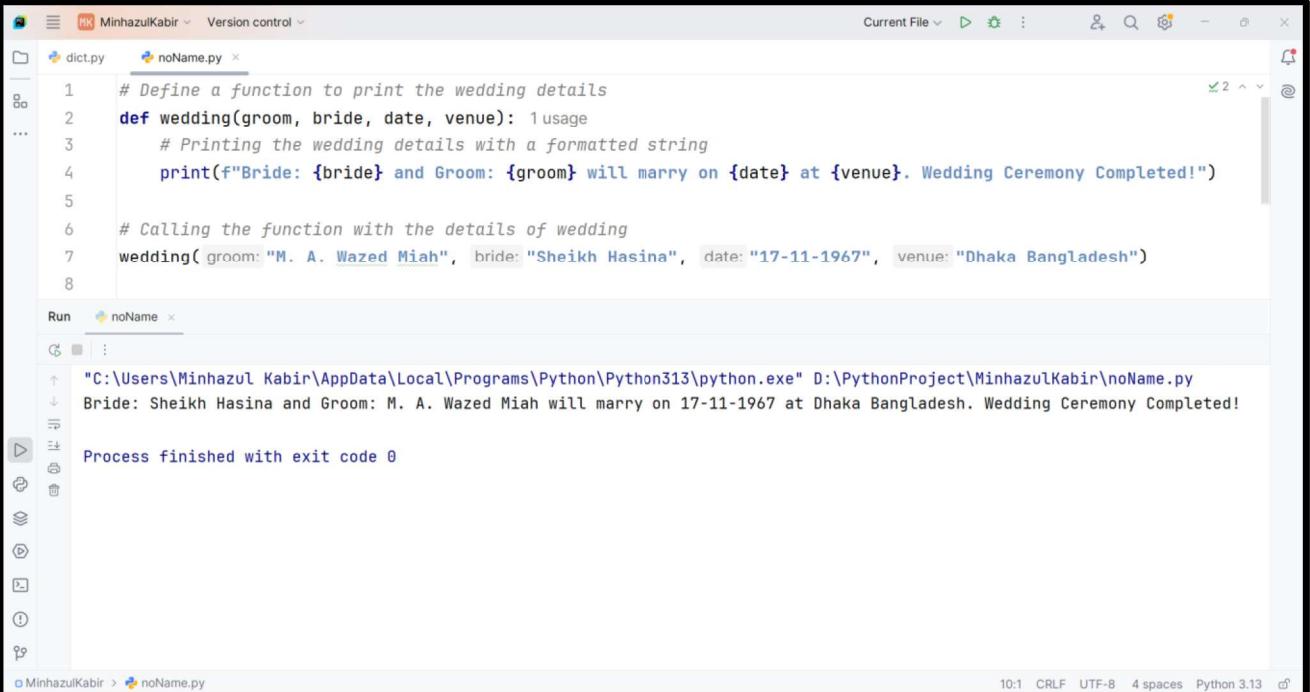
এখানে wedding ফাংশনটি বিভিন্ন parameter যেমন বর, কনে, তারিখ এবং স্থান নিয়ে কাজ করবে, এবং ফাংশনটির আউটপুট হবে একটি বিয়ের অনুষ্ঠান সম্পর্ক হওয়া।

function call:

```
# Calling the function with the details of wedding
wedding("M. A. Wazed Miah", "Sheikh Hasina", "17-11-1967", "Dhaka Bangladesh")
```

এটি আউটপুট দিবে:

Bride: Sheikh Hasina and Groom: M. A. Wazed Miah will marry on 17-11-1967 at Dhaka Bangladesh. Wedding Ceremony Completed!



```

MinhzulKabir Version control
dict.py noName.py

1 # Define a function to print the wedding details
2 def wedding(groom, bride, date, venue):
3     # Printing the wedding details with a formatted string
4     print(f"Bride: {bride} and Groom: {groom} will marry on {date} at {venue}. Wedding Ceremony Completed!")
5
6 # Calling the function with the details of wedding
7 wedding(groom: "M. A. Wazed Miah", bride: "Sheikh Hasina", date: "17-11-1967", venue: "Dhaka Bangladesh")
8

Run noName
C:\Users\Minhzul Kabir\AppData\Local\Programs\Python\Python313\python.exe D:\PythonProject\MinhzulKabir\noName.py
Bride: Sheikh Hasina and Groom: M. A. Wazed Miah will marry on 17-11-1967 at Dhaka Bangladesh. Wedding Ceremony Completed!

Process finished with exit code 0

10:1 CRLF UTF-8 4 spaces Python 3.13

```

define করার সময়ে function এর মধ্যে যে variable নাম লিখতে হয়, সে variable হচ্ছে parameter.

আর, function calling এর সময়ে যে value দেয়া হয় সে value হচ্ছে argument.

4. Exploring built-in functions.

নিচে Python এর কিছু সাধারণ built-in functions এর নাম এবং তাদের কাজ দেওয়া হল:

1. **abs():** কাজ: একটি সংখ্যার বিমূর্ত মান (absolute value) প্রদান করে।
2. **all():** কাজ: একটি iterable এর সব উপাদান True হলে True, অন্যথায় False প্রদান করে।
3. **any():** কাজ: একটি iterable এর যেকোনো উপাদান True হলে True, অন্যথায় False প্রদান করে।

4. **bin():** কাজ: একটি সংখ্যাকে বাইনারি স্ট্রিং হিসেবে রূপান্তর করে।
5. **bool():** কাজ: একটি মানকে True অথবা False তে রূপান্তরিত করে।
6. **bytearray():** কাজ: একটি নতুন bytearray অবজেক্ট তৈরি করে।
7. **bytes():** কাজ: একটি নতুন bytes অবজেক্ট তৈরি করে।
8. **callable():** কাজ: চেক করে যে একটি অবজেক্ট callable (function বা method হিসেবে ব্যবহারযোগ্য) কি না।
9. **chr():** কাজ: একটি Unicode কোড পয়েন্ট থেকে একটি চরিত্র রিটার্ন করে।
10. **class():** কাজ: একটি নতুন ক্লাস তৈরি করে।
11. **complex():** কাজ: একটি complex (যৌগিক) সংখ্যা তৈরি করে।
12. **delattr():** কাজ: একটি অবজেক্টের নির্দিষ্ট অ্যাট্রিবিউট মুছে ফেলে।
13. **dict():** কাজ: একটি নতুন dictionary অবজেক্ট তৈরি করে।
14. **dir():** কাজ: একটি অবজেক্টের বৈশিষ্ট্য ও পদ্ধতি তালিকা রিটার্ন করে।
15. **divmod():** কাজ: দুটি সংখ্যার ভাগফল এবং ভাগশেষ প্রদান করে।
16. **enumerate():** কাজ: একটি iterable থেকে ইন্ডেক্সহ উপাদান প্রদান করে।
17. **eval():** কাজ: একটি স্ট্রিং হিসাবে দেয়া এক্সপ্রেশনকে evaluate (মূল্যায়ন) করে।
18. **exec():** কাজ: একটি স্ট্রিং বা কোড ব্লক হিসেবে দেওয়া Python কোড execute করে।
19. **filter():** কাজ: একটি function প্রয়োগ করে iterable থেকে কিছু উপাদান ফিল্টার করে।
20. **float():** কাজ: একটি মানকে float (দশমিক সংখ্যা) তে রূপান্তরিত করে।
21. **format():** কাজ: স্ট্রিং ফর্ম্যাটিং করার জন্য ব্যবহৃত হয়।
22. **frozenset():** কাজ: একটি immutable set তৈরি করে।
23. **getattr():** কাজ: একটি অবজেক্টের নির্দিষ্ট অ্যাট্রিবিউটের মান রিটার্ন করে।
24. **globals():** কাজ: গ্লোবাল স্কোপের ডিকশনারি রিটার্ন করে।
25. **hasattr():** কাজ: একটি অবজেক্টে নির্দিষ্ট অ্যাট্রিবিউট আছে কিনা চেক করে।
26. **hash():** কাজ: একটি অবজেক্টের হ্যাশ মান প্রদান করে।
27. **help():** কাজ: একটি অবজেক্টের সম্পর্কে সহায়িকা (documentation) দেখায়।
28. **hex():** কাজ: একটি সংখ্যাকে হেক্সাডেসিমাল (ষাটকরা) স্ট্রিংয়ে রূপান্তরিত করে।
29. **id():** কাজ: একটি অবজেক্টের ইউনিক আইডি রিটার্ন করে।
30. **input():** কাজ: ব্যবহারকারীর কাছ থেকে ইনপুট নেয়।
31. **int():** কাজ: একটি মানকে পূর্ণসংখ্যা (integer) তে রূপান্তরিত করে।
32. **isinstance():** কাজ: চেক করে যে একটি অবজেক্ট নির্দিষ্ট ক্লাস বা ডেটা টাইপের ইনস্ট্যান্স কি না।
33. **issubclass():** কাজ: চেক করে যে একটি ক্লাস অন্য ক্লাসের সাবক্লাস কি না।
34. **iter():** কাজ: একটি iterable থেকে iterator তৈরি করে।

35. **len()**: কাজ: একটি অবজেক্টের দৈর্ঘ্য রিটার্ন করে (যেমন: তালিকা, স্ট্রিং, ইত্যাদি)।
36. **list()**: কাজ: একটি নতুন তালিকা (list) তৈরি করে।
37. **locals()**: কাজ: লোকাল স্কোপের ডিকশনারি রিটার্ন করে।
38. **map()**: কাজ: একটি function প্রতিটি উপাদানে প্রয়োগ করে একটি নতুন iterable তৈরি করে।
39. **max()**: কাজ: একটি iterable থেকে সবচেয়ে বড় মান রিটার্ন করে।
40. **memoryview()**: কাজ: একটি অবজেক্টের মেমরি ভিট রিটার্ন করে।
41. **min()**: কাজ: একটি iterable থেকে সবচেয়ে ছোট মান রিটার্ন করে।
42. **next()**: কাজ: একটি iterator থেকে পরবর্তী উপাদান রিটার্ন করে।
43. **object()**: কাজ: একটি নতুন object তৈরি করে।
44. **oct()**: কাজ: একটি সংখ্যাকে অকটাল (আটকরা) স্ট্রিংয়ে রূপান্তরিত করে।
45. **open()**: কাজ: একটি ফাইল খুলে।
46. **ord()**: কাজ: একটি চরিত্রের Unicode কোড পয়েন্ট প্রদান করে।
47. **pow()**: কাজ: একটি সংখ্যাকে একটি শক্তি (exponentiation) রূপে রূপান্তরিত করে।
48. **print()**: কাজ: কনসোলে আউটপুট প্রিন্ট করে।
49. **property()**: কাজ: একটি অ্যাট্রিবিউটকে getter, setter, বা deleter এর মাধ্যমে নিয়ন্ত্রণ করতে সাহায্য করে।
50. **range()**: কাজ: নির্দিষ্ট পরিসরে সংখ্যার একটি sequence তৈরি করে।
51. **repr()**: কাজ: একটি অবজেক্টের string রিপ্রেজেন্টেশন প্রদান করে।
52. **reversed()**: কাজ: একটি iterable এর উপাদানগুলি উল্টো করে রিটার্ন করে।
53. **round()**: কাজ: একটি সংখ্যা রাউন্ড করে নির্দিষ্ট সংখ্যক দশমিক পর্যন্ত।
54. **set()**: কাজ: একটি নতুন set তৈরি করে।
55. **setattr()**: কাজ: একটি অবজেক্টের নির্দিষ্ট অ্যাট্রিবিউটের মান সেট করে।
56. **slice()**: কাজ: একটি slice object তৈরি করে।
57. **sorted()**: কাজ: একটি iterable থেকে সাজানো (sorted) তালিকা রিটার্ন করে।
58. **staticmethod()**: কাজ: একটি স্ট্যাটিক মেথড তৈরি করে।
59. **str()**: কাজ: একটি মানকে স্ট্রিং (string) তে রূপান্তরিত করে।
60. **sum()**: কাজ: একটি iterable এর উপাদানগুলির যোগফল রিটার্ন করে।
61. **super()**: কাজ: সুপারক্লাসের মেথড বা অ্যাট্রিবিউট অ্যাক্সেস করতে সাহায্য করে।
62. **tuple()**: কাজ: একটি নতুন tuple তৈরি করে।
63. **type()**: কাজ: একটি অবজেক্টের টাইপ রিটার্ন করে।
64. **vars()**: কাজ: একটি অবজেক্টের dict রিটার্ন করে।
65. **zip()**: কাজ: একাধিক iterable থেকে tuple তৈরি করে।

এইগুলি Python এর কিছু গুরুত্বপূর্ণ built-in functions। Python ডকুমেন্টেশন থেকে আরও বিস্তারিত জানার জন্য আপনি দেখতে পারেন।

এদের মধ্যে অনেক function আমরা ইতিমধ্যেই ব্যবহার করে ফেলেছি এবং কাজ জানি, তারা কি কাজ করে।

3. Understanding recursion in Python.

Return Type এর উপর ভিত্তি করে function দুটি ধরনের হয়:

1. Return Type Function: যে ফাংশন কোনো মান (value) ফেরত দেয়, তাকে Return Type Function বলা হয়।

বিশেষত্ব: এই ফাংশনটি কাজ শেষ হওয়ার পর একটি নির্দিষ্ট value ফেরত দেয়, যা পরবর্তী কাজে ব্যবহার করা যেতে পারে।

2. Non-Return Type Function (বা Void Function): যে ফাংশন কোনো মান ফেরত দেয় না, তাকে Non-Return Type Function বা Void Function বলা হয়।

বিশেষত্ব: এই ফাংশনটি কোনো মান ফেরত না দিয়ে কেবল কিছু কাজ সম্পাদন করে (যেমন প্রিন্ট করা বা কোনো তথ্য পরিবর্তন করা), এবং ফলস্বরূপ কোনো ফলাফল ফিরে আসে না।

এখানে return type function এবং non-return type function এর মধ্যে পার্থক্য তুলে ধরা হলো:

দিক	Return Type Function	Non-Return Type Function
ফলাফল	মান (value) ফেরত দেয়।	কোনো মান ফেরত দেয় না।
উদ্দেশ্য	কোনো value পুনরঃদ্বারা বা ফেরত দেওয়া।	শুধুমাত্র কোনো কাজ সম্পাদন করা।
Return স্টেটমেন্ট	return কৌণ্যার্ড ব্যবহার হয়।	return কৌণ্যার্ড ব্যবহার হয় না বা None ফেরত দেয়।
ফলাফল ব্যবহারের স্থান	ফেরত পাওয়া মান অন্য কোথাও ব্যবহার করা যায়।	ফেরত পাওয়া মান ব্যবহার করা যায় না।
কাজের ধরন	মূলত তথ্য/value ফেরত দেওয়া।	প্রিন্ট করা বা কোনো কার্যক্রম সম্পাদন করা।

For more details:

<https://www.geeksforgeeks.org/deep-dive-into-parameters-and-arguments-in-python/?ref=roadmap>

<https://www.geeksforgeeks.org/python-return-statement/?ref=roadmap>

<https://www.geeksforgeeks.org/recursive-functions/?ref=roadmap>

return হচ্ছে variable/value. যার নির্দিষ্ট মান রয়েছে।

```
def minhaz():
    n=10
    p=20
    return n+p
print(type(minhaz()))
```

<class 'int'>

return না লিখকে কিছু ফেরত দিবে না। যার কোনো মান নাই।

```
def minhaz():
    n=10
    p=20
print(type(minhaz()))
```

<class 'NoneType'>

arguments এর মান পরিবর্তন হতে পারে। আমরা argument যা দিবো, সেরূপ উভর পাবো। এখন তুমি, নিচের 2 টি program এর output আমাকে দাও।

```
def print_sum(a, b):
    result = a + b
    print(result, end=" ") # This is a non-return type function
print(f"value of non return {print_sum(10,17)}")
print(f"value of non return {print_sum(20,13)}")
```

```
def add(a, b):
    result = a + b
    return result # This is a return type function
print(f"value of return {add(10,17)}")
print(f"value of return {add(20,13)}")
```

Recursive function হলো এমন একটি ফাংশন যা নিজেকেই কল (অথবা পুনরায় ব্যবহৃত) করে। এটি সাধারণত সমস্যাকে ছোট ছোট অংশে ভাগ করে সমাধান করতে ব্যবহৃত হয়। প্রতিটি রিকার্সিভ কল একটি বেস কেস (base case) দিয়ে শেষ হয়, যা পুনরাবৃত্তি বন্ধ করতে সাহায্য করে।

উদাহরণ: ধৰা যাক, আমৱা একটি সংখ্যা n এর ফ্যাক্টরিয়াল বেৱ কৱতে চাই। ফ্যাক্টরিয়াল হলো
 $n! = n \times (n-1) \times (n-2) \times \dots \times 1$

ফ্যাক্টরিয়াল গণনাৰ জন্য একটি রিকাৰ্সিভ ফাংশন নিচে দেওয়া হলো:

```
def factorial(n):
    # Base case: the factorial of 0 or 1 is 1
    if n == 0 or n == 1:
        return 1
    else:
        # Recursive case: n * factorial of (n-1)
        return n * factorial(n - 1)
```

কিভাবে কাজ কৱে:

- যখন n 0 বা 1 হয়, তখন ফাংশন 1 রিটোৰ্ন কৱে, যা বেস কেস।
- অন্যথায়, ফাংশন নিজেকে কল কৱে $n * \text{factorial}(n - 1)$ ।

উদাহরণ:

ধৰা যাক আমৱা $\text{factorial}(5)$ কল কৱি:

1. $\text{factorial}(5) \rightarrow 5 * \text{factorial}(4)$
2. $\text{factorial}(4) \rightarrow 4 * \text{factorial}(3)$
3. $\text{factorial}(3) \rightarrow 3 * \text{factorial}(2)$
4. $\text{factorial}(2) \rightarrow 2 * \text{factorial}(1)$
5. $\text{factorial}(1) \rightarrow 1$ (এটি বেস কেস)

ফিরে আসা মান:

- $\text{factorial}(1) = 1$
- $\text{factorial}(2) = 2 * 1 = 2$
- $\text{factorial}(3) = 3 * 2 = 6$
- $\text{factorial}(4) = 4 * 6 = 24$
- $\text{factorial}(5) = 5 * 24 = 120$

তাহলে, $\text{factorial}(5)$ এৱ ফলাফল হবে 120।

এভাবে, রিকাৰ্সিভ ফাংশন সমস্যাকে ছোট ছোট উপ-সমস্যায় ভাগ কৱে সমাধান কৱে।

HW. One-line loop বলতে পাইথনে কি বুৰা ? ব্যাখ্যা কৱো ।

5. Anonymous functions or lambda expressions.

return type function এর জমজ ভাই হচ্ছে lamda function.

Lambda function ব্যবহার করতে lambda কৌণ্ডার্ড ব্যবহার করা হয়। একটি লাম্বডা ফাংশন সাধারণত নিম্নলিখিত আকারে লেখা হয়:

```
lambda parameters: expression
```

এখানে:

- parameters: ফাংশনের ইনপুট (যেমন, a, b, ইত্যাদি)।
- expression: ফাংশনের return (এটি একটি এক্সপ্রেশন যা ইনপুটের উপর ভিত্তি করে গণনা করে)।

যেহেতু, lamda return type function সেজন্য lamda এর value কে চলকের মধ্যে সংরক্ষণ করতে হয়।

```
add_var = lambda a,b: a + b
print(add_var(33,10))
```

```
def add_var(a,b):
    return a+b
print(add_var(33,10))
```

এদের উভয়েরই output 43

লাম্বডা ফাংশন কেন ব্যবহার করা হয়?

- এটি ছোট ও দ্রুত ফাংশন তৈরির জন্য উপযোগী।
- যেখানে একটি একক কার্যক্রম (যেমন, যোগফল, গুণফল, তুলনা) সংজ্ঞায়িত করতে হয়, তখন এটি খুবই সুবিধাজনক।
- সাধারণ ফাংশন ব্যবহার করার চেয়ে অনেক সময় কোডের আকার কমাতে সাহায্য করে।

তবে, লাম্বডা ফাংশন শুধুমাত্র সিম্পল কার্যক্রমে ব্যবহার করা উচিত, কারণ এটি দীর্ঘ ও জটিল কোডের জন্য উপযুক্ত নয়।

lambda ফাংশনকে "anonymous function" বলা হয়, কারণ এটি একটি নামহীন ফাংশন। অর্থাৎ, lambda ফাংশনকে আপনি কোন নাম প্রদান করেন না (যেমন নির্যামিত ফাংশনে থাকে)। এর বদলে, এটি এক লাইনে সরাসরি একটি এক্সপ্রেশন প্রদান করে এবং তা কোন নাম ছাড়াই প্রয়োগ করা যায়। এই কারণে, lambda ফাংশনকে "anonymous" বা অজ্ঞাতনামা ফাংশন বলা হয়।

```
def myfunc(n):
    return lambda a : a * n
mydoubler = myfunc(5)
print(mydoubler(11))
```

ধাপ ১: myfunc ফাংশন

```
def myfunc(n):
    return lambda a : a * n
```

এখানে myfunc একটি ফাংশন যা একটি আর্গুমেন্ট n নেয় এবং একটি lambda ফাংশন রিটার্ন করে। এই lambda ফাংশনটি একটি আর্গুমেন্ট a নেয় এবং a * n রিটার্ন করে। অর্থাৎ, lambda a: a * n একটি ফাংশন তৈরি করছে যা a-কে n দ্বারা গুণ করবে।

ধাপ ২: mydoubler = myfunc(2)

```
mydoubler = myfunc(5)
```

এখানে myfunc(5) কল করা হয়েছে। এটি n = 5 প্যারামিটারসহ myfunc ফাংশনটি কল করবে। ফাংশনটি lambda a : a * 5 রিটার্ন করবে, অর্থাৎ এটি এমন একটি ফাংশন তৈরি করবে যা একটি সংখ্যাকে 5 দ্বারা গুণ করবে।

mydoubler নামের চলকটি এখন lambda a : a * 5 ফাংশনটি ধারণ করবে।

ধাপ ৩: mydoubler(11)

```
print(mydoubler(11))
```

এখানে mydoubler ফাংশনটি কল করা হয়েছে এবং আর্গুমেন্ট হিসেবে 11 পাঠানো হয়েছে। কারণ mydoubler ফাংশন হল lambda a : a * 5, এটি 11 কে 5 দ্বারা গুণ করবে। তাই $11 * 5 = 55$ হবে।

আউটপুট:

55

কী হচ্ছে এখানে?

1. myfunc(5) কল করার সময়, এটি lambda a: a * 5 একটি ফাংশন রিটার্ন করছে এবং mydoubler চলকটি তা গ্রহণ করছে।
2. যখন mydoubler(11) কল করা হচ্ছে, তখন এটি সেই lambda a: a * 5 ফাংশনটিকে চালু করছে এবং আর্গুমেন্ট 11 প্রদান করছে। ফলস্বরূপ, $11 * 5 = 55$ রিটার্ন হয়।

Closure/বন্ধ ধারণা:

এই উদাহরণে lambda ফাংশনটি একটি closure তৈরি করছে। অর্থাৎ, lambda ফাংশনটি যখন myfunc(5) থেকে রিটার্ন হয়, তখন এটি n = 5 ভেরিয়েবলটি মনে রাখে, এমনকি lambda ফাংশনটি যখন কল করা হয় তখনও। এটি n এর মানকে "remember" করে, এবং পরবর্তীতে যখন a পাস করা হয়, তখন n এর মান দিয়ে গুণ করা হয়।

noName.py ফাইল:

এই ফাইলটি একটি `minhaz(n)` নামক ফাংশন ধারণ করে। `minhaz(n)` ফাংশনটি একটি প্যারামিটার `n` গ্রহণ করে এবং `range(n)` ফাংশন ব্যবহার করে ০ থেকে `n-1` পর্যন্ত সংখ্যাগুলি প্রিন্ট করবে যখন `call` করা হবে। `end=" "` ব্যবহার করা হয়েছে, যার ফলে প্রতিটি সংখ্যা একটি স্পেস দিয়ে প্রিন্ট হবে, এবং নতুন লাইনে যাবে না।

```

def minhaz(n):
    for x in range(n):
        print(x, end=" ")

```

aTestFile.py ফাইল:

এখানে `noName.py` ফাইলটি `x` নামে ইমপোর্ট করা হয়েছে। এর মানে হলো, `x` দিয়ে `noName.py` ফাইলের ভিতরের সব ফাংশন এবং ভেরিয়েবল ব্যবহার করা যাবে।

এখানে `x.minhaz(10)` কল করা হয়েছে, যা `noName.py` ফাইলের `minhaz` ফাংশনটিকে কল করবে এবং আর্গুমেন্ট হিসেবে 10 পাঠাবে। `call` এর ফলস্বরূপ, `minhaz(10)` ০ থেকে ৯ পর্যন্ত সংখ্যাগুলো এক লাইনে স্পেস দিয়ে প্রিন্ট করবে।

```

import noName as x
x.minhaz(10)

```

Output in terminal:
C:\Users\Minhazul Kabir\AppData\Local\Programs\Python\Python313\python.exe" D:\PythonProject\MinhazulKabir\aTestFile.py
0 1 2 3 4 5 6 7 8 9
Process finished with exit code 0

Arbitrary Arguments, *args: আমরা এক চলকের মাধ্যমে list, tuple, set, dict ছাড়াও Arbitrary এর মাধ্যমে একাধিক value পাঠাতে পারি।

আপনি চাইলে `*args` এর মাধ্যমে অন্যান্য টাইপের ডাটা যেমন list, tuple, set, dict ইত্যাদিও পাঠাতে পারেন।

এটি উপকারী হতে পারে যখন আপনি জানেন না ফাংশনে কতগুলো আর্গুমেন্ট আসবে এবং আপনি সেই আর্গুমেন্টগুলিকে লুপ অথবা অন্যান্য পদ্ধতিতে প্রক্রিয়া করতে চান।

```
def minhaz(*n):
    for item in n:
        print(item)
minhaz("Emil", "Tobias", "Linus", 10, [1, 2, 3], {"key": "value"})
```

এখানে n চলকের মধ্যে ৫টি আলাদা ধরনের আর্গুমেন্ট পাঠানো হয়েছে, এবং প্রতিটি আর্গুমেন্ট আলাদা আলাদা ভাবে
প্রিন্ট করা হবে।

```
Emil
Tobias
Linus
10
[1, 2, 3]
{'key': 'value'}
```

এভাবে *args ব্যবহার করে একাধিক ভ্যালু পাঠানো এবং তাদের প্রক্রিয়া করা খুবই সহজ এবং নমনীয়।