

Class #07: File Handling and Exception Management

1. Opening, reading, append and writing files in Python.
2. Managing directories and creating files.
3. Difference between errors and exceptions.
4. Working with try-except and try-except-else statements.
5. Using try-except-finally statements. Assignment #7:

যখন আমরা কম্পিউটারে কোনো কাজ করতে চাই, যেমন একটি Microsoft Word ডকুমেন্ট লিখা, একটি মুভি চালু করা, অথবা কোনো ফাইল মুছে ফেলা, তখন প্রথমে সেই প্রোগ্রাম বা অ্যাপ্লিকেশনটি চালু করতে হয়। তারপর কাজ শেষ হলে, আমরা সেই প্রোগ্রামটি বন্ধ করি।

ঠিক তেমনি, Python-এ ফাইলের সাথে কাজ করতে গেলে, প্রথমে `open()` ফাংশন ব্যবহার করে ফাইলটি ওপেন করতে হয় এবং কাজ শেষে `close()` ফাংশন ব্যবহার করে ফাইলটি বন্ধ করতে হয়।

সহজ ভাষায়:

1. **ফাইল ওপেন করা:** যেমন আপনি কোনো প্রোগ্রাম ওপেন করেন, তেমনি Python-এ ফাইল ওপেন করার জন্য `open()` ফাংশন ব্যবহার করা হয়।
2. **কাজ শেষে ফাইল বন্ধ করা:** কাজ শেষ হলে, যেমন আপনি প্রোগ্রামটি বন্ধ করেন, তেমনি Python-এ ফাইলটির কাজ শেষ হলে `close()` ফাংশন ব্যবহার করে ফাইলটি বন্ধ করতে হয়।

```
Variable_name = open()
Variable_name.close()
```

1. Opening, reading, append and writing files in Python.

ফাইল ওপেনিং (Opening a File) Python-এ একটি ফাইল ওপেন করার জন্য `open()` ফাংশন ব্যবহার করা হয়। ফাইল ওপেন করার সময় আমরা ফাইলের জন্য একটি নির্দিষ্ট mode choose করি।

কিছু সাধারণ মোড:

1. '`r`' - **Read Mode:** এই মোডে ফাইলটি শুধুমাত্র পড়ার জন্য ওপেন করা হয়। ফাইলের কোনো কন্টেন্ট পরিবর্তন করা যায় না। যদি ফাইলটি না থাকে, তাহলে একটি ত্রুটি (Error) হবে।
2. '`a`' - **Append Mode:** এই মোডে ফাইলটি ওপেন করা হলে, ফাইলের পুরনো কন্টেন্টের শেষে নতুন তথ্য যোগ করা হয়। এখানে পুরনো কন্টেন্টে কোনো পরিবর্তন হয় না।

3. '**w**' – Write Mode: **w** মোডে ফাইল ওপেন করলে নতুন ফাইল তৈরি হয়, তবে যদি ফাইলটি আগে থেকেই বিদ্যমান থাকে, তাহলে তার পুরনো ফাইলটি স্থায়ীভাবে মুছে গিয়ে নতুন ফাইলে কন্টেন্ট লেখা হয়। অর্থাৎ, ফাইলটি আগে যেটা ছিল, সেটা পুরোপুরি মুছে গিয়ে নতুন লেখা হবে।
4. '**r+**' – Read-Write Mode: এই মোডে ফাইলটি ওপেন করা হলে, ফাইলটি পড়া এবং লেখা উভয় কাজই করা যাবে। অর্থাৎ, আপনি ফাইলটি পড়তে পারবেন এবং নতুন কন্টেন্টও লিখতে পারবেন।
5. '**x**' – Exclusive Creation Mode: **x** মোডে ফাইলটি শুধুমাত্র নতুনভাবে তৈরি করা হয় যদি পূর্বে ওই নামের কোনো ফাইল না থাকে। অর্থাৎ, যদি একই নামের ফাইল ইতিমধ্যেই বিদ্যমান থাকে, তবে এটি একটি ত্রুটি (Error) দেখাবে এবং নতুন ফাইল তৈরি হবে না। তবে, যদি ফাইলটি আগে না থাকে, তাহলে এটি নতুন ফাইল তৈরি করবে।

ধরুন, আপনি PyCharm এ একটি সাধারণ .txt ফাইল খুলেছেন। এবার আপনি সেখানে কিছু লেখা রাখতে চান। উদাহরণস্বরূপ, আপনি একটি ফাইল তৈরি করেছেন যার নাম **ratul.txt** এবং তার মধ্যে কিছু লেখা রেখেছেন:

The Beauty of Small Moments

True beauty often lies in life's quiet moments—the warmth of a hug, the sound of laughter, the colors of a sunset. In these simple instances, we find peace, connection, and joy. When we embrace the small wonders around us, life becomes a tapestry of quiet magic.

open() ফাংশন এবং close() ফাংশন আসলে কি?

open() ফাংশন এবং **close()** ফাংশন এর কাজ অনেকটা ফোনে অ্যাপ চালানোর মতো। ধরো তুমি ফোনে একটা ছবি তুলেছো, যদি সেই ছবি দেখতে চাও, তাহলে তুমি ফোনের photo gallery তে গিয়ে ছবিটা ওপেন করবে। এই কাজটাই হচ্ছে **open()** ফাংশন।

তুমি সারাদিন ফোনে অনেক অ্যাপ ব্যবহার করো—Facebook, Messenger, WhatsApp, Instagram, Camera app ইত্যাদি। যখন তুমি কোনো অ্যাপে টাচ করো, সেই অ্যাপটি ওপেন হয়। এইটাই হচ্ছে **open()** ফাংশন।

এখন, যদি তুমি WhatsApp থেকে Messenger এ চলে যাও, মানে WhatsApp বন্ধ করে দিলে। এরকম সময়েই **close()** ফাংশন কাজ করে। অর্থাৎ, এক অ্যাপ থেকে অন্য অ্যাপে যাওয়ার সময় পুরোনো অ্যাপটি **close()** হয়।

এভাবেই Python-এ **open()** এবং **close()** ফাংশন কাজ করে:

- ✿ **open()** ফাংশন দিয়ে ফাইল খুলে কাজ শুরু করা। এই **open()** ফাংশনটি দুটি argument নেয়: প্রথম argument হচ্ছে ফাইলের নাম এবং ঠিকানা (যেমন ratul.txt বা C:\Users\Minhazul Kabir\Desktop\ratul.txt), এবং দ্বিতীয় argument হচ্ছে মোড (যেমন 'r', 'w', 'r+', 'a')।
- ✿ এবং **close()** ফাংশন দিয়ে কাজ শেষে ফাইলটি বন্ধ করা।

'r' – Read Mode: এই মোডে ফাইলটি শুধুমাত্র পড়ার জন্য ওপেন করা হয়। ফাইলের কোনো কন্টেন্ট পরিবর্তন করা যায় না। read শব্দের প্রথম অক্ষর 'r' থেকেই এই মোডের নাম এসেছে। যদি ফাইলটি না থাকে, তাহলে একটি ত্রুটি (Error) হবে।

```
minhaz = open("ratul.txt", "r")
print(minhaz.read())
minhaz.close()
```

বা,

```
minhaz = open("ratul.txt", "r+")
print(minhaz.read())
minhaz.close()
```

'a' – Append Mode: append এর বাংলা অর্থ অতিরিক্তভাবে সংযুক্ত। এই মোডে ফাইলটি ওপেন করা হলে, লিখার মাধ্যমে ফাইলের পুরনো কন্টেন্টের শেষে নতুন তথ্য যোগ করা হয়। এখানে পুরনো কন্টেন্টে কোনো পরিবর্তন হয় না। যেহেতু append মোডে নতুন তথ্য যুক্ত করতে হয়, তাই এখানে write() ফাংশন ব্যবহার করা হয়। append() শব্দের প্রথম অক্ষর 'a' থেকেই এই মোডের নাম এসেছে।

```
minhaz = open("ratul.txt", "a")
minhaz.write("\nHappy New Year 2026")
minhaz.close()
```

আমরা ratul.txt open করলে দেখতে পাবো নিম্নরূপঃ

The Beauty of Small Moments

True beauty often lies in life's quiet moments—the warmth of a hug, the sound of laughter, the colors of a sunset. In these simple instances, we find peace, connection, and joy. When we embrace the small wonders around us, life becomes a tapestry of quiet magic.

Happy New Year 2026

'w' – Write Mode: এই মোডে ফাইলটি ওপেন করা হলে, ফাইলের পুরনো সমস্ত কন্টেন্ট মুছে গিয়ে নতুন কন্টেন্ট লেখা হয়। অর্থাৎ, ফাইলটি আগে যেটা ছিল, সেটা পুরোপুরি মুছে গিয়ে নতুন লেখা হবে। write শব্দের প্রথম অক্ষর 'w' থেকেই এই মোডের নাম এসেছে।

```
minhaz = open("ratul.txt", "w")
minhaz.write("This will overwrite existing all content.")
minhaz.close()
```

বা

```
minhaz = open("ratul.txt", "r+")
minhaz.write("This will overwrite existing all content.")
minhaz.close()
```

'r+' – Read-Write Mode: এই মোডে ফাইলটি ওপেন করা হলে, ফাইলটি পড়া এবং লেখা উভয় কাজই করা যাবে। অর্থাৎ, আপনি ফাইলটি পড়তে পারবেন এবং নতুন কন্টেন্টও লিখতে পারবেন।

```
minhaz = open("ratul.txt", "r+")
print(minhaz.read())
minhaz.write("\nThis will overwrite existing all content.")
minhaz.close()
```

'x' – Exclusive Creation Mode: **exclusive** এর বাংলা অর্থ একচেটিয়া বা বাধাদায়ক। এই মোডে ফাইলটি শুধুমাত্র নতুনভাবে তৈরি করা হয় যদি পূর্বে ওই নামের কোনো ফাইল না থাকে। অর্থাৎ, যদি একই নামের ফাইল ইতিমধ্যেই বিদ্যমান থাকে, তবে এটি একটি ত্রুটি (Error) দেখাবে এবং নতুন ফাইল তৈরি হবে না। তবে, যদি ফাইলটি আগে না থাকে, তাহলে এটি একটি নতুন ফাইল তৈরি করবে। এই মোডটি কাজ করবে তখন, যখন নিশ্চিত করতে হয় যে নির্দিষ্ট নামের ফাইলটি আগে থেকে নেই এবং ফাইলটি শুধুমাত্র নতুনভাবে তৈরি হবে।

```
minhaz = open("ratul.txt", "x")
minhaz.close()
```

উপসংহারণ

আমরা কম্পিউটার থেকে একটি ফাইলে ডাবল ক্লিক করে তা পড়তে পারি। ডাবল ক্লিক করার পর, আমরা সেখানে কিছু অতিরিক্ত লেখা যোগ করতে পারি, অথবা ফাইলের সব কন্টেন্ট মুছে নতুন করে লিখতে পারি।

এভাবে, Python প্রোগ্রামিংয়ের মাধ্যমে আমরা একই কাজ করতে পারি। Python দিয়ে ফাইল ওপেন করা, সেখানে নতুন তথ্য যোগ করা, বা ফাইলের পুরনো কন্টেন্ট মুছে নতুন কিছু লেখা—এ সব কিছুই করা সম্ভব।

2. Managing directories and creating files.

আমরা যেটাকে সাধারণ ভাষায় "ফোল্ডার" বলি, প্রোগ্রামিংয়ে সেটাকে "ডিরেক্টরি" (Directory) বলা হয়। আসলে, ফোল্ডার এবং Directory একই জিনিস। যেমন আমরা কম্পিউটার বা ফোনে ফোল্ডার তৈরি করতে পারি, তেমনি পাইথন প্রোগ্রামিং দিয়েও ফোল্ডার বা ডিরেক্টরি তৈরি করা যায়।

পাইথনে os লাইব্রেরি একটি স্ট্যান্ডার্ড লাইব্রেরি, যা অপারেটিং সিস্টেমের সাথে যোগাযোগ করতে ব্যবহৃত হয়।

"OS" এর পূর্ণরূপ হলো "Operating System" এর মাধ্যমে আমরা ফাইল এবং ডিরেক্টরি সম্পর্কিত বিভিন্ন কাজ করতে পারি, যেমন ফোল্ডার তৈরি করা, ফাইলের তথ্য দেখা, ফাইল মুছে ফেলা, এবং অন্যান্য অনেক কাজ।

<https://docs.python.org/3/library/os.html#module-os>

os লাইব্রেরি কিছু গুরুত্বপূর্ণ ফিচার:

- ❖ ফাইল ও ডিরেক্টরি সম্পর্কিত কাজ:

- ❖ os.mkdir("folder_name"): একটি নতুন ডিরেক্টরি তৈরি করে।
- ❖ os.remove("file_name"): একটি ফাইল মুছে ফেলে।
- ❖ os.listdir(): একটি ডিরেক্টরির মধ্যে সব ফাইল ও ফোল্ডারের তালিকা দেখায়।

- ❖ সিস্টেমের তথ্য:

- ❖ os.getcwd(): বর্তমান কাজের ডিরেক্টরি (current working directory) দেখায়।
- ❖ os.environ: পরিবেশ পরিবর্তনশীল (environment variables) অ্যাক্সেস করতে ব্যবহৃত হয়।

- ❖ পথ সম্পর্কিত কাজ:

- ❖ os.path.join(): বিভিন্ন পথের অংশ একত্রিত করে একটি সম্পূর্ণ পথ তৈরি করে।

অতএব, os মডিউল Python-এ এমন সব কার্যকলাপ করতে সাহায্য করে যা অপারেটিং সিস্টেমের সাথে সম্পর্কিত।

1. ডিরেক্টরি/ফোল্ডার তৈরি করা (Creating Directories/folder): Python-এ os লাইব্রেরি ব্যবহার করে আমরা নতুন ডিরেক্টরি/ফোল্ডার তৈরি এবং পুরানো ডিরেক্টরি/ফোল্ডার মুছে ফেলতে পারি।

নতুন ফোল্ডার তৈরি করা (Create a new folder):

নতুন একটি ডিরেক্টরি বা ফোল্ডার তৈরি করতে os.mkdir() ফাংশন ব্যবহার করা হয়। এখানে mkdir শব্দটি make directory এর সংক্ষেপ, যার মাধ্যমে একটি নতুন ডিরেক্টরি বা ফোল্ডার তৈরি করা হয়। এর গঠন হলো:

os.mkdir("folder_name"), যেখানে "folder_name" হচ্ছে ফোল্ডারের নাম যা আপনি তৈরি করতে চান।

```
import os
os.mkdir("harryPotter")
```

ব্যাখ্যা:

- ❖ import os: এটি os মডিউলটি ব্যবহার করার জন্য।
- ❖ os.mkdir("harryPotter"): এটি "harryPotter" নামে একটি নতুন ফোল্ডার তৈরি করবে।

ফোল্ডার মুছে ফেলা (delete folder):

একটি ডিরেক্টরি বা ফোল্ডার মুছে ফেলতে `os.rmdir()` ফাংশন ব্যবহার করা হয়। এখানে `rmdir` শব্দটি `remove directory` এর সংক্ষেপ, যার মাধ্যমে একটি ফোল্ডার স্থায়ীভাবে মুছে ফেলা হয়। এর গঠন হলো:

```
os.rmdir("folder_name"), যেখানে "folder_name" হচ্ছে সেই ফোল্ডারের নাম যা আপনি মুছে ফেলতে চান।
```

```
import os # Importing the os module
os.rmdir("harryPotter") # removing the directory named "harryPotter"
```

ব্যাখ্যা:

- ✿ `import os`: এটি `os` মডিউলটি ব্যবহার করার জন্য।
- ✿ `os.rmdir("harryPotter")`: এটি "harryPotter" নামের ফোল্ডারটি মুছে ফেলবে।

2. ডিরেক্টরি পরিচালনা (Managing Directories):

Python-এ `os` লাইব্রেরি ব্যবহার করে আপনি বিভিন্ন ডিরেক্টরি/ফোল্ডারের মধ্যে যেতে পারেন, বর্তমান কাজের ডিরেক্টরি চেক করতে পারেন বা ডিরেক্টরি সরাতে পারেন। `os.listdir()` দিয়ে একটি ডিরেক্টরির সমস্ত ফাইলের তালিকা পাওয়া।

```
import os
print(os.listdir())
```

- 1) `import os` – এটি `os` মডিউলটি ব্যবহার করার জন্য `import` করা হয়, যা অপারেটিং সিস্টেম সম্পর্কিত কাজগুলো করতে সাহায্য করে।
- 2) `os.listdir()` – এটি বর্তমান ডিরেক্টরির (ফোল্ডারের) ভিতরের সব ফাইল ও ফোল্ডারের নামের তালিকা প্রদান করে।
- 3) `print()` – এটি সেই তালিকা কনসোলে প্রদর্শন করে।

3. ফাইল তৈরি করা (Creating Files):

ফাইল মুছে ফেলতে `os.remove()` ফাংশন ব্যবহার করা হয়। এখানে `remove` শব্দটির অর্থ মুছে ফেলা, যার মাধ্যমে একটি ফাইল স্থায়ীভাবে মুছে ফেলা হয়। এর গঠন হলো:

```
os.remove("file_name"), যেখানে "file_name" হচ্ছে সেই ফাইলের নাম যা আপনি মুছে ফেলতে চান।
```

```
import os
os.remove("ratul.txt")
```

- ✿ `import os`: এটি `os` মডিউলটি ব্যবহার করার জন্য।
- ✿ `os.remove("ratul.txt")` – এটি "ratul.txt" নামের ফাইলটি মুছে ফেলবে। যদি এই নামের ফাইলটি বর্তমান ডিরেক্টরিতে না থাকে, তবে এটি একটি ত্রুটি (Error) দেখাবে।

3. Difference between errors and exceptions.

1. Error: Error সাধারণত সিস্টেম বা পরিবেশের ভিতরে ঘটে যা প্রোগ্রামের নিয়ন্ত্রণের বাইরে থাকে। এগুলি সাধারণত গুরুতর সমস্যা এবং এগুলি প্রোগ্রাম চলাকালীন কিছু করতে পারে না। উদাহরণস্বরূপ, মেমরি কম হলে বা হার্ডওয়্যার সমস্যা হলে একটি Error তৈরি হতে পারে।

উদাহরণ: ধরা যাক, আপনি একটি বড় সফটওয়্যার চালাচ্ছেন এবং আপনার সিস্টেমের মেমরি শেষ হয়ে গেছে। এই অবস্থায়, প্রোগ্রাম চালানো সম্ভব হবে না, কারণ মেমরি নেই, এটি একটি Error।

2. Exception: Exception এমন সমস্যা যা কোডের ভিতরে ঘটে, কিন্তু এটি সাধারণত ধরা এবং হ্যান্ডেল করা যায়। যখন আপনি কিছু ভুল করেন বা অপ্রত্যাশিত কিছু ঘটে, তখন এটি Exception তৈরি করে। আপনি এই Exception এর জন্য কোডে ব্যবস্থা নিতে পারেন।

উদাহরণ: ধরা যাক, আপনি এমন একটি কোড লিখেছেন যেখানে শূন্য দিয়ে ভাগ করার চেষ্টা করা হচ্ছে, যা একটি সাধারণ Exception তৈরি করবে।

মূল পার্থক্য:

- ✚ Error: সিস্টেম বা পরিবেশগত সমস্যা, যা প্রোগ্রাম চালানো সম্ভব করে না। (যেমন `MemoryError`)
- ✚ Exception: কোডের মধ্যে ঘটে, তবে এটি হ্যান্ডেল করা যায়। (যেমন `ZeroDivisionError`, `FileNotFoundException`)

সাধারণত:

- ✚ Error সাধারণত কোডের ভুল নয়, বরং সিস্টেমের বা পরিবেশের সমস্যা।
- ✚ Exception হলো কোডের ভুল বা অপ্রত্যাশিত পরিস্থিতি, যেগুলি প্রোগ্রাম চালানো অবস্থায় সংশোধন করা যেতে পারে।

4. Working with try-except and try-except-else statements.

ধরা যাক, আপনি আপনার ফোনে কাউকে কল করতে যাচ্ছেন। প্রথমে আপনি `try` করছেন কলটি করার জন্য, অর্থাৎ ফোনের ডায়ালার খুলে কল দিতে চেষ্টা করছেন। যদি কোনো সমস্যা না হয়, তাহলে কলটি সফলভাবে চলে যাবে। তবে, যদি কোনো সমস্যা থাকে, কল যাবে না এবং তখন `except block` কাজ করবে এবং সমস্যাটি সমাধান করার চেষ্টা করবে।

ফোনে কল না যাওয়ার বিভিন্ন কারণ থাকতে পারে, যেমন:

- **নেটওয়ার্ক সমস্যা:** যদি নেটওয়ার্ক না থাকে, তবে আপনি হয়তো এক জায়গা থেকে অন্য জায়গায়, যেমন বারান্দায় বা ওপেন জায়গায় গিয়ে চেষ্টা করতে পারেন। এটি নেটওয়ার্কের সমস্যা সমাধান করতে সাহায্য করতে পারে।
- **ব্যালেন্স সমস্যা:** যদি আপনার ফোনে ব্যালেন্স না থাকে, তবে আপনি জায়গা পরিবর্তন করেও কিছু করতে পারবেন না। সেক্ষেত্রে, আপনাকে দোকানে গিয়ে ব্যালেন্স রিচার্জ করতে হবে।
- **ফোনে চার্জ না থাকা:** যদি আপনার ফোনে চার্জ না থাকে, তাহলে শুধু জায়গা পরিবর্তন করে কিছু হবে না। আপনাকে ফোনটি চার্জ করতে বিদ্যুতের সাথে সংযোগ দিতে হবে।

- ফোন হ্যাং হওয়া: যদি ফোন হ্যাং করে, তাহলে জায়গা পরিবর্তন করে কিছু হবে না। আপনাকে ফোনটি রিস্টার্ট করতে হবে এবং তারপর আবার চেষ্টা করতে হবে।

এই পুরো প্রক্রিয়াটি **try-except** ব্লকের মতো। আপনি প্রথমে কিছু কাজ বা কোড চালানোর চেষ্টা করছেন (যেমন কল করা), এবং যদি কোনো সমস্যা (ক্রটি) ঘটে, তখন **except block** ব্যবহার করে সেই সমস্যার সমাধান করার চেষ্টা করছেন।

গঠনঃ

try:

```
# Place the code you want to test here
# If any error occurs in this code, the except block will execute
except ExceptionType1:
    # Here you will handle the first type of exception
except ExceptionType2:
    # Here you will handle the second type of exception
except ExceptionType3:
    # Here you will handle the third type of exception
...
...
...
```

"try" দিয়ে শুরু হয়ে, "except" দিয়ে একাধিক ব্লক লেখা যায়, ঠিক যেমন "if", "elif" এর মতো।

ধরা যাক, আপনি ব্যবহারকারীর ইনপুট থেকে দুটি সংখ্যা নিয়ে তাদের ভাগফল বের করতে চান, কিন্তু এখানে কিছু ক্রটি ঘটতে পারে (যেমন, শূন্য দিয়ে ভাগ করা বা অযথা ইনপুট দেয়া):

```
try:
    num1 = int(input("Enter the first number: "))
    num2 = int(input("Enter the second number: "))
    result = num1 / num2
except ValueError:
    print("The value you entered is not a number!")
except ZeroDivisionError:
    print("Division by zero is not possible!")
except Exception as e_minhaz:
    print(f"An unknown error occurred: {e_minhaz}")
```

ব্যাখ্যা:

1. **try:** আপনি প্রথমে দুটি সংখ্যা ইনপুট নিয়ে তাদের ভাগফল বের করার চেষ্টা করছেন।

2. **ValueError:** যদি ব্যবহারকারী সংখ্যা ছাড়া অন্য কিছু (a,b,c,d) ইনপুট দেয়, তবে এটি ধরা হবে এবং "The value you entered is not a number!" বার্তা প্রিন্ট হবে।
3. **ZeroDivisionError:** যদি দ্বিতীয় সংখ্যাটি শূন্য হয়, তবে এটি "Division by zero is not possible!" বার্তা প্রিন্ট করবে।
4. **Exception:** অন্যান্য যেকোনো অজানা ক্ষটি যদি ঘটে, তবে এটি "An unknown error occurred:" বার্তা সহ ক্ষটির বিস্তারিত দেখাবে।

এইভাবে, একাধিক **except** ব্লক ব্যবহার করে আমরা বিভিন্ন ধরনের ক্ষটি আলাদাভাবে হ্যান্ডেল করতে পারি।

গঠন: পাইথনে **try**, **except**, এবং **else** ব্লকগুলি একসাথে ব্যবহার করা হয়

try:

```
# Place the code you want to test here
# If any error occurs in this code, the except block will execute
```

except ExceptionType1:

```
# Here you will handle the first type of exception
```

except ExceptionType2:

```
# Here you will handle the second type of exception
```

except ExceptionType3:

```
# Here you will handle the third type of exception
```

...

...

...

else:

```
# If the code in the try block runs successfully without any error, this will be executed
```

Python এ **try** ব্লক দিয়ে শুরু হয়, তারপর প্রয়োজন অনুযায়ী একাধিক **except** ব্লক থাকতে পারে, এবং সর্বশেষে একটি **else** ব্লক ব্যবহার করা হয়। এটি **if**, **elif**, **else** এর মতো কাজ করে।

অর্থাৎ, প্রথমে **try** দিয়ে সম্ভাব্য এক্সেপশন (ক্ষটি) আটকানো হয়, তারপর **except** দিয়ে সেসব এক্সেপশন হ্যান্ডেল করা হয়, এবং যদি কোনো এক্সেপশন না ঘটে, তবে **else** ব্লকটি কার্যকর হয়।

ধরা যাক, আপনি ব্যবহারকারীর ইনপুট থেকে দুটি সংখ্যা নিয়ে তাদের ভাগফল বের করতে চান, কিন্তু এখানে কিছু ক্ষটি ঘটতে পারে (যেমন, শূন্য দিয়ে ভাগ করা বা অযথা ইনপুট দেয়া):

try:

```
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))
result = num1 / num2
```

except ValueError:

```

print("The value you entered is not a number!")

except ZeroDivisionError:
    print("Division by zero is not possible!")

except Exception as e_minhaz:
    print(f"An unknown error occurred: {e_minhaz}")

else:
    print("Result:", result)

```

ব্যাখ্যা:

1. **try:** আপনি প্রথমে দুটি সংখ্যা ইনপুট নিয়ে তাদের ভাগফল বের করার চেষ্টা করছেন।
2. **except ValueError:** যদি ব্যবহারকারী ইনপুট হিসেবে সংখ্যার পরিবর্তে অন্য কিছু দেন, তবে ValueError হবে এবং "The value you entered is not a number!" বার্তা দেখাবে।
3. **except ZeroDivisionError:** যদি দ্বিতীয় সংখ্যাটি শূন্য হয়, তাহলে ZeroDivisionError হবে এবং "Division by zero is not possible!" বার্তা দেখাবে।
4. **except Exception:** অন্য কোনো অজানা ক্রটি ঘটলে, তা "An unknown error occurred:" বার্তা সহ দেখানো হবে।
5. **else:** যদি try ব্লকে কোনো ক্রটি না ঘটে, তাহলে else ব্লকটি কাজ করবে এবং Result প্রিন্ট হবে।

এই লিঙ্ক দেখো: https://www.w3schools.com/python/python_try_except.asp

5. Using try-except-finally statements.

Python-এ try, except, else, এবং finally statements গুলি ব্যবহার করে error handling করা হয়, যা আপনাকে কোডে potential errors বা exceptions handle করার সুযোগ দেয়। এছাড়া, কোডটি error-free হলে আপনি else ব্লকেও কিছু কোড execute করতে পারেন, এবং finally ব্লকটি নিশ্চিত করবে যে কোডটি error হোক বা না হোক, শেষ পর্যন্ত কিছু action হবে।

1. try ব্লক:

- ✿ এখানে আপনি সেই কোডটি রাখবেন, যা error বা exception সৃষ্টি করতে পারে।
- ✿ যদি এই ব্লকে কোনো exception ঘটে, তবে Python কোড execution থামিয়ে except ব্লকে চলে যাবে।

2. except ব্লক:

- ✿ try ব্লকে যদি কোনো exception ঘটে, তবে সেই exception ধরার জন্য except ব্লক ব্যবহৃত হয়। error না হলে else এ চলে যাবে। এখানে আপনি exception সম্পর্কে সিদ্ধান্ত নিতে পারেন এবং error handle করতে পারেন।

3. else ব্লক:

- else ব্লকটি try ব্লক সফলভাবে execute হলে, অর্থাৎ কোনো exception না ঘটলে execute হবে। এটি ব্যবহার করে আপনি সেই কোড লিখতে পারেন, যেটি শুধুমাত্র error-free কোড execution হলে চলবে।

4. finally ব্লক:

- এই ব্লকটি সর্বদা execute হবে, error হোক বা না হোক। এটি সাধারণত resource clean-up বা file closing এর মতো কাজের জন্য ব্যবহৃত হয়। finally ব্লকটি try-except block এর পরেও কাজ করে।

Syntax:

```
try:
    # Code that may raise an exception
except <ExceptionType>:
    # What will happen if an exception is caught
else:
    # What will happen if no exception occurs
finally:
    # This code will always execute
```

উদাহরণ:

```
try:
    # Code that may raise an error
    num1 = int(input("Enter a number: "))  # Input from user
    num2 = int(input("Enter another number: "))
    result = num1 / num2  # Division
except ValueError:
    # If an invalid input (non-integer) is given
    print("Invalid input! Please enter a valid number.")
except ZeroDivisionError:
    # If an attempt is made to divide by zero
    print("Error: Cannot divide by zero!")
else:
    # If no exception occurs, meaning the division is successful
    print(f"The result is {result}.")
finally:
    # This block will always execute
    print("This block always runs.")
```

Output:**Scenario 1:**

```
Enter a number: 10
```

```
Enter another number: 2
```

```
The result is 5.0.
```

```
This block always runs.
```

Scenario 2 (invalid input):

```
Enter a number: abc
```

```
Invalid input! Please enter a valid number.
```

```
This block always runs.
```

Scenario 3 (division by zero):

```
Enter a number: 10
```

```
Enter another number: 0
```

```
Error: Cannot divide by zero!
```

```
This block always runs.
```

ব্যাখ্যা:

- try ব্লক:** এখানে কোডে এমন কিছু রয়েছে যা exception সৃষ্টি করতে পারে, যেমন ValueError বা ZeroDivisionError।
- except ব্লক:** try ব্লকের মধ্যে exception ঘটলে তা ধরার জন্য এটি ব্যবহৃত হয়। আমরা দুটি exception ধরেছি: ValueError এবং ZeroDivisionError।
- else ব্লক:** যদি কোনো exception না ঘটে, তবে else ব্লকটি execute হবে এবং result print হবে।
- finally ব্লক:** এটা সর্বদা execute হবে, error হোক বা না হোক। এটি সাধারণত resource clean-up বা file closing এর জন্য ব্যবহৃত হয়।

উপসংহার:

- ✿ **try:** কোড যেটা error সৃষ্টি করতে পারে।
- ✿ **except:** exception ঘটলে যা করতে হবে।
- ✿ **else:** exception না ঘটলে যা করতে হবে।
- ✿ **finally:** যেকোনো পরিস্থিতিতেই কোডটি execute হবে।

নিম্নলিখিত কোডে try, except, else, এবং finally ব্লক ব্যবহার করা হয়েছে:

```
try:
    lemon = open("fardin.txt", "r")
    print(lemon.read())
except FileNotFoundError:
    print("The file 'fardin.txt' was not found.")
except IOError:
    print("An error occurred while reading the file.")
else:
    print("File read successfully.")
finally:
    try:
        lemon.close() # Attempt to close the file
        print("File closed.")
    except NameError:
        print("File was not opened, so it cannot be closed.")
```

1. **try ব্লক:** try ব্লকটি কোডের সেই অংশ যেখানে আপনি এমন কিছু কার্যকলাপ করতে চান যা ত্রুটি সৃষ্টি করতে পারে। এর মধ্যে কোনো সমস্যা হলে, সেই ত্রুটিকে ধরতে except ব্লক ব্যবহার করা হবে। **উদাহরণ:**

```
try:
    lemon = open("fardin.txt", "r") # ফাইল খোলার চেষ্টা
    print(lemon.read()) # ফাইলের কন্টেন্ট পড়ার চেষ্টা
```

এখানে try ব্লকের মধ্যে ফাইল খোলার এবং ফাইলের কন্টেন্ট পড়ার চেষ্টা করা হচ্ছে। যদি ফাইলটি না পাওয়া যায় বা পড়ার সময় অন্য কোনো সমস্যা হয়, তখন সেই ত্রুটির জন্য except ব্লকটি কার্যকর হবে।

2. **except ব্লক:** except ব্লকটি try ব্লকের ত্রুটির মোকাবিলা করে। যখন try ব্লকের কোনো ত্রুটি ঘটে, তখন except ব্লকটি সেসব ত্রুটি ক্যাচ করে এবং আপনি সেই ত্রুটির জন্য একটি নির্দিষ্ট বার্তা বা হ্যান্ডলিং করতে পারেন। **উদাহরণ:**

```
except FileNotFoundError:
    print("The file 'fardin.txt' was not found.")
except IOError:
    print("An error occurred while reading the file.")
```

এখানে দুইটি except ব্লক রয়েছে:

✿ প্রথমটি FileNotFoundError ত্রুটির জন্য, যা তখন ঘটে যখন ফাইলটি খোঁজার সময় পাওয়া না যায়।

- ✿ দ্বিতীয়টি IOError ক্রটির জন্য, যা ফাইল খোলার বা পড়ার সময় অন্য কোনো ইনপুট/আউটপুট সমস্যা (যেমন ডিক্ষ সমস্যা) ঘটলে ঘটে।

3. **else ব্লক:** else ব্লকটি শুধুমাত্র তখন কার্যকর হয় যখন try ব্লকের কোনো ক্রটি না ঘটে। অর্থাৎ, try ব্লকটি সফলভাবে সম্পন্ন হলে, তখন else ব্লকটি কার্যকর হবে। **উদাহরণ:**

```
else:
```

```
    print("File read successfully.")
```

এখানে যদি ফাইলটি সঠিকভাবে পড়া যায় (যেমন ক্রটি ছাড়াই), তবে else ব্লকটি চালু হবে এবং "File read successfully." এই বার্তাটি প্রিন্ট হবে।

4. **finally ব্লক:** finally ব্লকটি সবসময় কার্যকর হয়, তা কোন ক্রটি ঘটুক বা না ঘটুক। এটি এমন কার্যকলাপের জন্য ব্যবহৃত হয় যা শেষমেশ সম্পন্ন করতে হবে, যেমন ফাইল বন্ধ করা বা অন্যান্য ক্লিনআপ কাজ। **উদাহরণ:**

```
finally:
```

```
try:
```

```
    lemon.close() # ফাইল বন্ধ করার চেষ্টা
```

```
    print("File closed.")
```

```
except NameError:
```

```
    print("File was not opened, so it cannot be closed.")
```

- ✿ এখানে finally ব্লকটি সবসময় কার্যকর হবে, কিন্তু lemon.close() কে try ব্লকে রাখা হয়েছে কারণ যদি ফাইলটি খুলতে না পারে, তাহলে lemon ভেরিয়েবলটি সংজ্ঞায়িত হবে না এবং NameError হতে পারে।
- ✿ যদি ফাইলটি সফলভাবে না খোলা হয় (ফাইলটি খুঁজে না পাওয়া যায়), তবে except NameError ব্লকটি কার্যকর হবে এবং একটি বার্তা দেখাবে যে ফাইলটি খোলা হয়নি, তাই এটি বন্ধ করা যাবে না।

সারাংশ:

- **try:** কোডের সেই অংশ যেখানে আপনি ক্রটি হতে পারে এমন কাজটি করতে চান।
- **except:** যখন try ব্লকে ক্রটি ঘটে, তখন এটি সেই ক্রটি ক্যাচ করে এবং প্রয়োজনীয় ব্যবস্থাপনা করে।
- **else:** যদি try ব্লকটি সফলভাবে চলে (অর্থাৎ কোনো ক্রটি না ঘটে), তখন এটি কার্যকর হবে।
- **finally:** এটি সবসময় কার্যকর হবে, তা try ব্লকে কোনো ক্রটি ঘটুক বা না ঘটুক, এবং এটি সাধারণত ক্লিনআপ (যেমন ফাইল বন্ধ করা) করার জন্য ব্যবহৃত হয়।

এই চারটি ব্লক একত্রে ব্যবহার করা হলে, আপনার কোডটি ক্রটি হ্যান্ডলিং এবং ক্লিনআপ কার্যকলাপের জন্য খুবই শক্তিশালী হয়ে ওঠে।