Date parsing in Python

Date parsing in Python is a common operation when working with temporal data coming from strings, such as timestamps in log files, form inputs, or data exported from external systems.

1. Using datetime.strptime

The `datetime` module from Python's standard library provides the `strptime` function, which allows you to convert a string into a `datetime` object by specifying the expected format:

```
from datetime import datetime

d_str = "2025-07-23 14:30:00"
format = "%Y-%m-%d %H:%M:%S"

d = datetime.strptime(d_str, format)
print(d)
```

2. Flexible Parsing with dateutil.parser

To avoid specifying the format manually, you can use the `dateutil` module, which provides a more intelligent parser:

```
from dateutil import parser
```

```
d = parser.parse("23 July 2025, 14:30")
print(d)
```

This approach is useful when the date format may vary or is not known in advance.

3. Error Handling

It’s good practice to handle potential exceptions during parsing, especially when working with external input:

```
from datetime import datetime

try:
    d = datetime.strptime("2025-07-23", "%d/%m/%Y")
except ValueError as e:
    print(f"Parsing error: {e}")
```

4. Parsing with Pandas

If you’re working with tabular datasets, `pandas` provides a very convenient function to convert entire columns into datetime objects:

```
import pandas as pd

df = pd.DataFrame({"date_str": ["2025-07-23", "2025-08-01"]})
df["date"] = pd.to_datetime(df["date_str"])
print(df)
```

Conclusion

Python offers several tools for date parsing. `datetime.strptime` is precise but requires explicit formats, while `dateutil` and `pandas` provide more flexible and powerful alternatives, especially in data analysis contexts.