11 - Python Standard Library

Intro to Python Standard Libraries

- Python Standard Libraries provide a rich set of modules and functions.
- They help perform various tasks without requiring external packages.
- This chapter covers some of the most commonly used libraries.

dataclasses

The **dataclasses** module provides a decorator and functions for creating data classes.

Data classes are a way to define classes primarily storing state without writing boilerplate code.

Reference: https://docs.python.org/3/library/dataclasses.html

Useful for:

- Simplifying class definitions.
- Automatically generating special methods like **__init__()**, **__repr__()**, and **__eq__()**

dataclasses

When to use?

- When you need to create classes for storing data without implementing boilerplate code.
- For creating simple data containers like configs, records, etc.

Some use cases:

- Configuration objects in applications
- Storing records from databases
- Payloads for API request or response

```python
from dataclasses import dataclass

@dataclass
class Person:
    name: str
    age: int

p1 = Person("Alice", 30)

print(p1)  # Output: Person(name='Alice', age=30)
```

datetime

The datetime module supplies classes for manipulating dates and times.

Reference: https://docs.python.org/3/library/datetime.html

Useful for:

- Supporting date and time arithmetic.
- Provides extra functionality for formatting and parsing dates and times.

datetime

When to use?

- When working with date and time data, such as timestamps, scheduling, or logging.
- For date and time calculations, formatting, and parsing.

Some use cases:

- Scheduling applications.
- Logging systems with timestamps.
- Date and time formatting and parsing in reports and data processing

```
from datetime import datetime


now = datetime.now()

print(now)  # Output: 2024-07-28 12:34:56.789123


formatted_date = now.strftime("%Y-%m-%d %H:%M:%S")

print(formatted_date)  # Output: 2024-07-28 12:34:56
```

functools

The functools module provides higher-order functions that act on or return other functions.

Reference: https://docs.python.org/3/library/functools.html

Useful for:

- Simplifying functional programming.
- Includes extra tools like **lru_cache**, **partial**, **reduce**, etc..

functools

When to use?

- When you need to enhance or compose functions.
- For memoization, function partial application, or performing reductions.

Some use cases:

- Caching expensive function calls.
- Creating reusable function templates.
- Performing cumulative operations on data collections.

```
from functools import lru_cache


@lru_cache(maxsize=None)
def fibonacci(n):
    if n < 2:
        return n
    return fibonacci(n-1) + fibonacci(n-2)


print(fibonacci(10))  # Output: 55
```

itertools

The itertools module provides functions for creating iterators for efficient looping.

Reference: https://docs.python.org/3/library/itertools.html

Useful for:

- Ideal for handling combinatorial problems and infinite sequences.
- tools like **count**, **cycle**, **chain**, and **combinations**.

itertools

When to use?

- When you need to efficiently iterate over data collections.
- For creating complex iteration patterns or handling combinatorial problems.

Some use cases:

- Infinite sequence generation.
- Combining multiple iterators.
- Creating permutations and combinations for algorithmic problems.

```
from itertools import count

for i in count(start=10, step=2):
    if i > 20:
        break
    print(i)  # Output: 10, 12, 14, 16, 18, 20
```

logging

The logging module provides a flexible framework for emitting log messages from Python programs.

Reference: https://docs.python.org/3/library/logging.html

Useful for:

- Allows for different logging levels like **DEBUG**, **INFO**, **WARNING**, **ERROR**, etc.
- Supports logging to different destinations (console, file, etc.).

logging

When to use?

- When you need to track events that happen during program execution.
- For debugging, monitoring, and alerting purposes in applications.

Some use cases:

- Debugging and error tracking in development.
- Monitoring application performance and behavior in production.
- Generating audit trails and logs for compliance and analysis.

```python
import logging


logging.basicConfig(level=logging.DEBUG)
logging.debug("This is a debug message")
logging.info("This is an info message")
logging.warning("This is a warning message")
logging.error("This is an error message")
logging.critical("This is a critical message")
```

timeit

The timeit module provides a simple way to time small bits of Python code.

Reference: https://docs.python.org/3/library/timeit.html

Useful for:

- Performance testing and optimization.
- Measuring execution time of code snippets accurately.

timeit

When to use?

- When you need to measure the execution time of small code snippets.
- For comparing the performance of different implementations.

Some use cases:

- Benchmarking algorithms and functions.
- Identifying performance bottlenecks in code.
- Comparing different coding approaches for efficiency.

```
import timeit


code_to_test = """

result = sum(range(100))

"""

execution_time = timeit.timeit(code_to_test, number=1000)

print(f"Execution time: {execution_time} seconds")
```

pickle

The pickle module provides a simple way to convert a Python object into a byte stream and vice versa.

Reference: https://docs.python.org/3/library/pickle.html

Useful for:

- Ideal for saving and loading Python objects to and from a file.
- Quick and easy way to serialize/deserialize Python-specific objects.
- Supports complex data types like custom classes, functions, etc.

pickle

When to use?

- Saving Model States: Save complex data structures for later use.
- Session Persistence: Store the state of an application to resume later.
- Data Caching: Save results of expensive computations to avoid recalculating them.

Some Disadvantages:

- Not secure against untrusted data, as it can execute arbitrary code during deserialization.
- Pickle is Python-specific, so it may not be suitable for cross-language data exchange.

```python
import pickle


# Sample Python object (dictionary)
data = {'name': 'Alice', 'age': 25}


# Serialize to a file
with open('data.pkl', 'wb') as f:
    pickle.dump(data, f)
# Deserialize from the file
with open('data.pkl', 'rb') as f:
    loaded_data = pickle.load(f)


print(loaded_data)  # Output: {'name': 'Alice', 'age': 25}
```

Conclusion

- Python’s standard libraries offer robust and versatile tools for various tasks.
- Using these libraries can save time and effort in developing complex functionality.
- We’ve covered some of the most commonly used libraries:
  - dataclasses
  - datetime
  - functools
  - itertools
  - logging
  - timeit
  - pickle

When using a new library (whether that’s built-in or 3rd party), always refer to the documentation for reference.