Understanding pprint in Python

A Student-Friendly Guide

1. What Is pprint?

`pprint` (Pretty Print) is a built-in Python module that displays complex or nested data structures in a readable, formatted way. It is especially useful when working with:

- Nested dictionaries
- Lists of dictionaries
- JSON-like data
- Debugging complex program output

You can import it using:

```python
from pprint import pprint
```

or:

```python
import pprint
```

2. Why Use pprint Instead of print?

The standard `print()` function may output everything on one long line. `pprint` improves readability by:

- Adding indentation
- Breaking long lines
- Formatting nested structures
- Optionally sorting dictionary keys

Example:

```python
from pprint import pprint

data = {
    "name": "Alice",
    "scores": [95, 88, 74],
    "details": {"age": 20, "city": "London"}
}

pprint(data)
```

3. Core Functions

3.1 `pprint(object, **options)`

Prints an object in a pretty, human-readable format.

```
pprint(data)
```

Useful optional parameters:

- `indent=` number of spaces
- `width=` maximum line width
- `depth=` limits nesting
- `compact=` tries to use fewer lines
- `sort_dicts=` sorts keys if True

Example:

```
pprint(data, indent=4, width=60)
```

3.2 `pformat(object, **options)`

Returns the pretty-printed content as a string, instead of printing it.

```
from pprint import pformat

text = pformat(data)
print(text)
```

This is ideal for logging.

4. Writing Pretty Output to a File

When you see:

```
log.write(pformat(data))
```

This writes the formatted data into whatever file you opened.

Example:

```
from pprint import pformat

data = {"a": 1, "b": {"c": 2}}

with open("output.txt", "w") as log:
    log.write(pformat(data))
```

This saves the formatted text into `output.txt`.

5. The PrettyPrinter Class (Reusable Printer)

You can create a reusable pretty-printer with fixed formatting settings:

```
import pprint

printer = pprint.PrettyPrinter(indent=2, width=50)
```

You can then use it multiple times:

```
printer.pprint({"x": 1, "y": 2})
printer.pprint(["a", "b", "c"])
printer.pprint({"nested": {"a": {"b": 123}}})
```

Why use a reusable printer? - Ensures consistent formatting across your program
- Saves time — no need to repeat parameters
- Cleaner, easier-to-maintain code

Example with pformat:

```
formatted = printer.pformat(data)
print(formatted)
```

6. Quick Reference Table

Feature	**Description**
`pprint()`	Prints formatted output
`pformat()`	Returns formatted string
`indent`	Controls indentation
`width`	Line wrap width
`depth`	Limits nested printing
`compact`	Fewer lines if possible
`sort_dicts`	Sort keys alphabetically

7. Example: Before vs After pprint

Without pprint:

```
print(data)
```

Output (hard to read):

```
{'name': 'Alice', 'scores': [95, 88, 74], 'details': {'age': 20,
'city': 'London'}}
```

With pprint:

```
{'details': {'age': 20, 'city': 'London'},
 'name': 'Alice',
 'scores': [95, 88, 74]}
```

Much clearer and student-friendly.

8. Summary

- `pprint` is designed to make complex data readable.
- `pformat` lets you store or log formatted text.
- A reusable `PrettyPrinter` simplifies repeated formatting.
- Ideal for debugging, teaching, and working with nested data.