1. Iterators: The "One-at-a-Time" Rule

In Python, an **iterator** is an object that allows you to traverse through all the elements of a collection (like a list) one by one.

- **How it works:** It uses two magic methods: __iter__() (to initialize) and __next__() (to get the next value).
- **Why use it?** It saves memory because it doesn't load the whole list at once; it just remembers where it left off.

Python

```
my_list = [1, 2, 3]
my_iter = iter(my_list)

print(next(my_iter)) # Output: 1
print(next(my_iter)) # Output: 2
```

2. Generators: The Memory Savers

Generators are a simpler way to create iterators. Instead of return, they use the **yield** keyword.

- **The Difference:** A function with return stops completely. A function with yield **pauses** its state and can be resumed later.
- **Use case:** Reading a massive 10GB file. You don't want to load 10GB into RAM; you use a generator to read it line by line.

Python

```
def count_up_to(n):
  count = 1
  while count <= n:
    yield count
    count += 1
```

```
counter = count_up_to(3)
for num in counter:
  print(num) # 1, 2, 3
```

---

3. Decorators: The "Gift Wrappers"

A decorator allows you to add new functionality to an existing function without changing its source code. Think of it like adding sprinkles to an existing ice cream cone.

- **Syntax:** Uses the @ symbol above a function.
- **Use case:** Logging, timing how long a function takes, or checking if a user is logged in.

Python

```
def my_decorator(func):
  def wrapper():
    print("Something is happening before the function.")
    func()
    print("Something is happening after the function.")
  return wrapper

@my_decorator
def say_hello():
  print("Hello!")

say_hello()
```

---

4. Custom Exceptions: Personalize Your Errors

Sometimes, standard Python errors like ValueError aren't specific enough. You can create your own error types to make your code more readable.

- **How:** Create a class that inherits from the built-in Exception class.

Python

```
class BelowAgeError(Exception):
    """Raised when the input age is less than 18"""
    pass

age = 15
if age < 18:
    raise BelowAgeError("You are too young for this club!")
```

---

5. Logging vs. Printing

Beginners use print(), but professionals use logging.

Feature	**print()**	**logging**
Purpose	Quick debugging.	Recording events over time.
Control	Shows everything.	You can set levels (Info, Warning, Error).
Destination	Only the console.	Files, databases, or remote servers.

Pro Tip: Never use print() in production code. Use logging.info() or logging.error() so you can track what happened days later in a log file.

---

6. Data Warehouse vs. Data Mart

In the world of Big Data, we need specific places to store information for analysis.

Data Warehouse (DW)

A **Data Warehouse** is a large, central repository of data collected from many different sources across an entire company.

- **Scale:** Huge.
- **User:** Data Scientists and Corporate Analysts.

- **Focus:** The whole business (Sales + HR + Inventory + Marketing).

Data Mart

A **Data Mart** is a simple, smaller version of a Data Warehouse that focuses on a **single department** or subject.

- **Scale:** Small/Specific.
- **User:** Specific teams (e.g., just the Finance team).
- **Focus:** A single slice of the business.

Quick Comparison Table

Feature	**Data Warehouse**	**Data Mart**
Scope	Corporate-wide (Integrated)	Single department (Specific)
Data Source	Many sources	Few sources
Size	100GB to Terabytes	Often < 100GB
Setup Time	Months or Years	Weeks