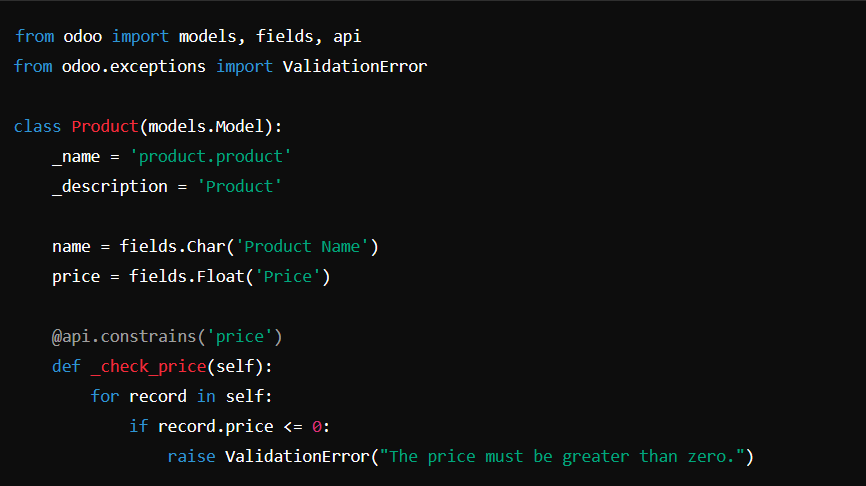
**1. Python Constraints**

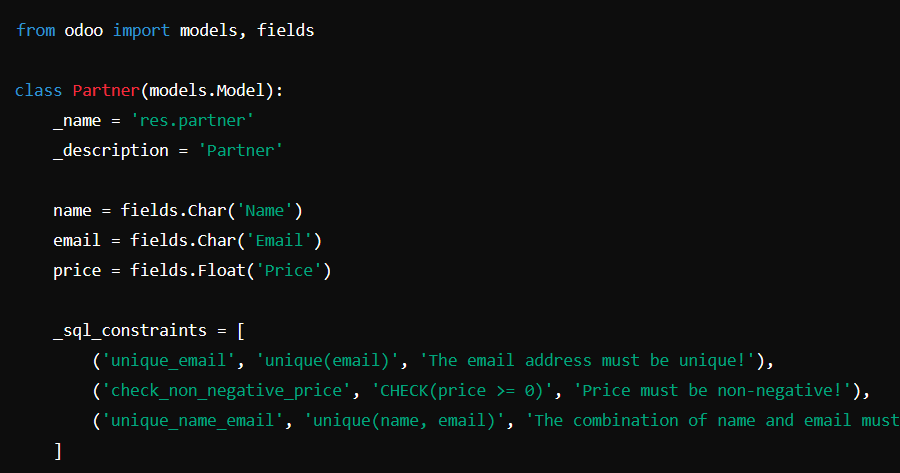
Python constraints are defined using the **@api.constrains** decorator in Odoo. This allows you to apply logic before a record is saved to the database.

****

* **@api.constrains('price'):** This decorator/method triggere whenever the price field is changed.

**2. SQL Constraints**

SQL constraints are applied directly to the database and are enforced at the database level. You can define these constraints in the model's \_\_table\_\_ attribute or in the fields definition.



**Key Differences:**

| **Feature** | **Python Constraints** | **SQL Constraints** |
| --- | --- | --- |
| **Location** | Defined in Python code using @api.constrains decorator. | Defined in the model’s \_sql\_constraints attribute. |
| **Execution** | Applied during the ORM operations (create, write) in Python code. | Applied directly by the database when data is inserted or updated. |
| **Flexibility** | Can use complex logic (e.g., check fields in other models, call functions). | Limited to SQL syntax and simple conditions (e.g., CHECK, UNIQUE). |
| **Error Handling** | Can raise custom ValidationError with detailed messages. | Raises a database-specific error (e.g., CheckViolation), which is handled by Odoo. |
| **Enforcement** | Enforced by the Odoo ORM during object lifecycle. | Enforced directly by the database engine, even outside Odoo. |
| **Database Impact** | No direct impact on the database schema. | Direct impact on the database schema (e.g., creating unique indexes, check constraints). |
| **Performance** | Slower since the validation occurs within Python. | Faster as the validation is handled directly by the database engine. |