🐰 ふわふわ大福店で学ぶPython継承完全ガイド

うさうさ店長と一緒に、継承を完全マスター！

📝 この記事について

こんにちは！ふわふわ大福店のうさうさ店長です🐰

この記事では、Pythonの**継承**について、実際に動くコードと一緒に学んでいきます。

✨ この記事で学べること

- ✅ **単一継承**（親 → 子）
- ✅ **階層継承**（親 → 子 → 孫）
- ✅ **多重継承**（親A + 親B → 子）
- ✅ それぞれの使い分け

🎯 対象読者

- Pythonの基本文法は理解している
- クラスの基礎は知っている
- 継承をもっと深く理解したい

それでは、一緒に学んでいきましょう！

📚 目次

継承とは何か

📖 継承の定義

継承 = 既存のクラスの機能を引き継いで、新しいクラスを作ること

例えば、こんなイメージです：

```
基本の大福店（親） → プレミアム大福店（子）

親の機能:
- 在庫管理
- 販売処理
- 情報表示

子の機能:
- 親の機能すべて ← 継承！
- VIP会員管理 ← 新しく追加
- プレミアム販売 ← 新しく追加
```

🎯 継承の3つのパターン

Pythonには3種類の継承パターンがあります：

パターン	構造	図	推奨度
単一継承	親 → 子	A → B	★★★★★
階層継承	親 → 子 → 孫	A → B → C	★★★☆☆
多重継承	親A,B → 子	A,B → C	★★☆☆☆

それぞれ詳しく見ていきましょう！

単一継承 - 最もシンプルな形

📖 単一継承とは

単一継承 = 1つの親クラスから1つの子クラスへ

最もシンプルで、最も推奨される継承の形です。

💻 コード例：基本店舗 → プレミアム店舗

ステップ1: 親クラスを作る

```python
```

```python
class DaifukuShop:
    """大福店の基本クラス（親クラス）"""

    def __init__(self, owner_name, stock):
        """
        コンストラクタ

        引数:
            owner_name: 店長の名前
            stock: 初期在庫数
        """
        # インスタンス変数の初期化
        self.owner_name = owner_name  # 店長名を保存
        self.stock = stock  # 在庫数を保存
        self.sold = 0  # 累計販売数を0で初期化

        print(f"🏪 {self.owner_name}店長の基本店舗を開店")

    def sell(self, quantity):
        """大福を販売する"""
        # 在庫チェック
        if quantity > self.stock:
            print(f"❌ 在庫不足！")
            return False

        # 在庫を減らす
        self.stock -= quantity
        self.sold += quantity

        print(f"💰 {quantity}個販売（残り: {self.stock}個）")
        return True

    def show_info(self):
        """店舗情報を表示"""
        print(f"\n{'='*50}")
        print(f"🏪 {self.owner_name}店長の店")
        print(f"📦 在庫: {self.stock}個")
        print(f"💰 累計販売: {self.sold}個")
        print(f"{'='*50}\n")
```

ステップ2: 子クラスを作る（継承）

python

```python
class PremiumDaifukuShop(DaifukuShop):
    """
    プレミアム大福店（子クラス）

    DaifukuShopを継承 ← ここが継承のポイント！
    """

    def __init__(self, owner_name, stock, vip_count):
        """
        コンストラクタ

        引数:
            owner_name: 店長名
            stock: 初期在庫
            vip_count: VIP会員数（子クラス固有）
        """
        # super()で親クラスのコンストラクタを呼ぶ
        # これで owner_name, stock, sold が初期化される
        super().__init__(owner_name, stock)

        # 子クラス固有の変数
        self.vip_count = vip_count

        print(f"✨ プレミアム機能追加（VIP: {vip_count}名）")

    def sell_premium(self, quantity):
        """
        プレミアム販売（子クラス固有のメソッド）

        通常価格の1.5倍で販売
        """
        # 親クラスのsellメソッドを呼ぶ
        result = super().sell(quantity)

        if result:
            price = quantity * 225  # 150円 × 1.5
            print(f"✨ プレミアム価格: ¥{price}")

        return result

    def show_info(self):
        """
        情報表示（オーバーライド）

        親のメソッドを拡張してVIP情報を追加
        """
```

```
        # 親のshow_infoを呼ぶ
        super().show_info()

        # 子クラス固有の情報を追加
        print(f"👑 VIP会員: {self.vip_count}名")
        print(f"{'='*50}\n")
```

ステップ3: 使ってみる

```python
# 親クラスのインスタンス
basic = DaifukuShop("うさうさ", 20)
basic.sell(5)
basic.show_info()

# 子クラスのインスタンス
premium = PremiumDaifukuShop("もちもち", 30, 5)
premium.sell(3)  # 親から継承
premium.sell_premium(2)  # 子クラス固有
premium.show_info()  # オーバーライド
```

📊 単一継承の構造

```
DaifukuShop（親）
 ├── owner_name
 ├── stock
 ├── sold
 ├── sell()
 └── show_info()
   ↓ 継承
PremiumDaifukuShop（子）
 ├── （親から継承）
 │   ├── owner_name
 │   ├── stock
 │   ├── sold
 │   └── sell()
 ├── （子クラス固有）
 │   ├── vip_count
 │   ├── sell_premium()
 │   └── show_info()（オーバーライド）
```

💡 単一継承のポイント

✅ 良い点

- シンプルで理解しやすい
- メソッド解決が明確
- デバッグしやすい
- 最も推奨される形

⚠️ 注意点

- `super().__init__()`を忘れずに呼ぶ
- is-a関係が成り立つか確認
  - ◯ 「プレミアム店は大福店である」
  - × 「車はエンジンである」

🎯 使うべき場面

- 基本機能を少し拡張したい
- シンプルに保ちたい
- 迷ったらまずこれ！

階層継承 - 親から子、子から孫へ

📖 階層継承とは

階層継承 = 継承が何世代にもわたって続く形

親 → 子 → 孫と、段階的に機能を追加していきます。

💻 コード例：基本店舗 → プレミアム → VIP専門店

ステップ1: 孫クラスを作る

```python
```

```
class VIPDaifukuShop(PremiumDaifukuShop):
    """
    VIP専門店（孫クラス）

    継承チェーン:
    DaifukuShop → PremiumDaifukuShop → VIPDaifukuShop
    """

    def __init__(self, owner_name, stock, vip_count, concierge_count):
        """
        コンストラクタ

        引数:
            owner_name: 店長名
            stock: 初期在庫
            vip_count: VIP会員数
            concierge_count: コンシェルジュ数（孫クラス固有）
        """
        # 親（PremiumDaifukuShop）のコンストラクタを呼ぶ
        # これで祖父（DaifukuShop）も自動的に初期化される
        super().__init__(owner_name, stock, vip_count)

        # 孫クラス固有の変数
        self.concierge_count = concierge_count

        print(f"👑 VIP専門店機能追加（コンシェルジュ: {concierge_count}名）")

    def sell_vip_exclusive(self, quantity):
        """
        VIP限定販売（孫クラス固有のメソッド）

        超高級品：プレミアム価格の2倍
        """
        # 親のsell_premiumを呼ぶ
        result = super().sell_premium(quantity)

        if result:
            price = quantity * 450  # 225円 × 2
            print(f"👑 VIP限定価格: ¥{price}")
            print(f"📞 コンシェルジュがお届け")

        return result

    def show_info(self):
        """情報表示（さらにオーバーライド）"""
        # 親のshow_infoを呼ぶ
```

```
        super().show_info()

        # 孫クラス固有の情報
        print(f"👔 コンシェルジュ: {self.concierge_count}名")
        print(f"{'='*50}\n")
```

ステップ2: 使ってみる

```
python

# 孫クラスのインスタンス
vip = VIPDaifukuShop("ぴょんぴょん", 40, 10, 3)

# 祖父クラスから継承
vip.sell(2)

# 親クラスから継承
vip.sell_premium(3)

# 孫クラス固有
vip.sell_vip_exclusive(1)

# すべての情報を表示
vip.show_info()
```

📊 階層継承の構造

```
DaifukuShop（祖父）
 ├── owner_name
 ├── stock
 └── sell()
   ↓ 継承
PremiumDaifukuShop（父）
 ├── 祖父から継承した全部
 ├── vip_count（追加）
 └── sell_premium()（追加）
   ↓ 継承
VIPDaifukuShop（子＝孫）
 ├── 父・祖父から継承した全部
 ├── concierge_count（追加）
 └── sell_vip_exclusive()（追加）
```

💡 階層継承のポイント

✅ 良い点

- 段階的に機能を追加できる
- カテゴリー分けが明確
- 再利用性が高い

⚠️ 注意点

- **階層は2-3階層まで**が推奨
- 深すぎると複雑になる
- `super()`の連鎖に注意

❌ 悪い例：深すぎる階層

```python
# これは深すぎる（6階層）
class A:
    pass
class B(A):
    pass
class C(B):
    pass
class D(C):
    pass
class E(D):
    pass
class F(E):  # 深すぎる！
    pass
```

🎯 使うべき場面

- 段階的に特化させたい
- カテゴリーが明確
- 2-3階層で収まる

多重継承 - 複数の親から

📖 多重継承とは

多重継承 = 複数の親クラスから同時に継承

異なる機能を持つ複数のクラスを組み合わせます。

💻 コード例：店舗 + オンライン + 配達 + 会員

ステップ1: 4つの親クラスを作る

python

```python
# 親A: 店舗機能
class ShopBase:
    """基本的な店舗機能"""

    def __init__(self, owner_name, stock):
        self.owner_name = owner_name
        self.stock = stock
        self.sold = 0
        print(f"🏪 店舗機能: {owner_name}店長")

    def sell(self, quantity):
        """基本販売"""
        if quantity > self.stock:
            return False
        self.stock -= quantity
        self.sold += quantity
        print(f"💰 {quantity}個販売")
        return True


# 親B: オンライン機能
class OnlineServiceMixin:
    """オンライン注文機能"""

    def __init__(self):
        self.online_orders = 0
        print(f"🌐 オンライン機能: 起動")

    def receive_online_order(self, customer, quantity):
        """オンライン注文受付"""
        self.online_orders += 1
        print(f"🌐 オンライン注文 #{self.online_orders}")
        print(f"   顧客: {customer}様")
        print(f"   数量: {quantity}個")


# 親C: 配達機能
class DeliveryServiceMixin:
    """配達サービス機能"""

    def __init__(self):
        self.deliveries = 0
        print(f"🚚 配達機能: 起動")

    def deliver(self, address):
        """配達実行"""
```

```
        self.deliveries += 1
        print(f"🚚 配達 #{self.deliveries}: {address}へ")


# 親D: 会員機能
class MembershipServiceMixin:
    """会員サービス機能"""

    def __init__(self):
        self.members = []
        print(f"💳 会員機能: 起動")

    def register_member(self, name):
        """会員登録"""
        self.members.append(name)
        print(f"💳 {name}様を登録（会員数: {len(self.members)}名）")
```

ステップ2: 4つを統合した子クラスを作る

python

```
class HybridDaifukuShop(ShopBase, OnlineServiceMixin,
                        DeliveryServiceMixin, MembershipServiceMixin):
    """
    ハイブリッド店舗（多重継承）

    4つの親から機能を継承:
    - ShopBase（店舗）
    - OnlineServiceMixin（オンライン）
    - DeliveryServiceMixin（配達）
    - MembershipServiceMixin（会員）
    """

    def __init__(self, owner_name, stock):
        """
        コンストラクタ

        すべての親を初期化する必要がある
        """
        print(f"\n{'='*60}")
        print(f"🌟 ハイブリッド店舗を起動")
        print(f"{'='*60}")

        # 各親クラスを初期化
        ShopBase.__init__(self, owner_name, stock)
        OnlineServiceMixin.__init__(self)
        DeliveryServiceMixin.__init__(self)
        MembershipServiceMixin.__init__(self)

        print(f"✅ すべての機能が利用可能")
        print(f"{'='*60}\n")

    def process_order(self, customer, quantity, address):
        """
        統合注文処理

        すべての親の機能を組み合わせて使う
        """
        print(f"\n注文処理: {customer}様")

        # 1. オンライン注文（親B）
        self.receive_online_order(customer, quantity)

        # 2. 販売（親A）
        if self.sell(quantity):
            # 3. 配達（親C）
            self.deliver(address)
```

```
        # 4. 会員登録（親D）
        if customer not in self.members:
            self.register_member(customer)

        print(f"✅ 注文完了\n")
    else:
        print(f"❌ 注文失敗\n")
```

ステップ3: 使ってみる

```
python

# ハイブリッド店舗を作成
hybrid = HybridDaifukuShop("ふわふわ", 50)

# すべての機能を使った注文処理
hybrid.process_order("田中太郎", 5, "東京都渋谷区")
hybrid.process_order("佐藤花子", 3, "東京都新宿区")

# 統計表示
print(f"📊 統計情報")
print(f"在庫: {hybrid.stock}個")
print(f"販売: {hybrid.sold}個")
print(f"オンライン注文: {hybrid.online_orders}件")
print(f"配達回数: {hybrid.deliveries}回")
print(f"会員数: {len(hybrid.members)}名")
```

📊 多重継承の構造

```
ShopBase   OnlineService   DeliveryService   MembershipService
   ↘          ↓               ↓               ↙
              HybridDaifukuShop
              （すべての機能を持つ）
```

💡 多重継承のポイント

✅ 良い点

- 独立した機能を組み合わせられる
- コードの再利用性が高い
- ミックスインパターンに最適

⚠️ 注意点

- **複雑になりやすい**
- MRO（メソッド解決順序）を理解する必要
- ダイヤモンド継承に注意
- すべての親を初期化する

🎯 使うべき場面

- 独立した複数の機能が必要
- ミックスインパターン
- 機能の組み合わせ

❌ 使うべきでない場面

- 関連が深すぎる親同士
- 5つ以上の親から継承
- シンプルな拡張で済む場合

3つの継承パターン比較

📊 一覧比較表

項目	単一継承	階層継承	多重継承
構造	親 → 子	親 → 子 → 孫	親A,B,C → 子
親の数	1つ	1つずつ	複数
階層の深さ	2階層	3階層以上	2階層
複雑さ	⭐ 簡単	⭐⭐ 中程度	⭐⭐⭐ 複雑
推奨度	★★★★★	★★★☆☆	★★☆☆☆
使用頻度	90%	9%	1%

🎯 視覚的な違い

```
【単一継承】
  親
  ↓
  子

シンプル・明確


【階層継承】
  祖父
```

```
  ↓
 父
  ↓
 子
```

縦に深い

```
【多重継承】
親A  親B  親C
 ↘  ↓  ↙
   子
```

横に広い

💡 使い分けフローチャート

Q: どの継承を使うべき？

```
├── 基本機能を少し拡張したい？
│   └── YES → 単一継承
│
├── 段階的に機能を追加したい？
│   └── YES → 階層継承（2-3階層まで）
│
└── 複数の独立した機能を組み合わせたい？
    └── YES → 多重継承（慎重に）
```

実践：継承の使い分け

✅ 単一継承を使うべき場面

python

```python
# 良い例1: シンプルな拡張
class User:
    pass

class AdminUser(User):  # 管理者はユーザーである
    pass


# 良い例2: 機能の追加
class Document:
    pass

class PDFDocument(Document):  # PDF文書は文書である
    pass
```

✅ 階層継承を使うべき場面

```python
# 良い例: 段階的な特化
class Employee:  # 従業員
    pass

class Manager(Employee):  # マネージャーは従業員である
    pass

class Director(Manager):  # 役員はマネージャーである
    pass

# 3階層で収まっている✓
```

✅ 多重継承を使うべき場面

```python
```

```python
# 良い例: 独立した機能の組み合わせ
class Saveable:  # 保存機能
    pass

class Printable:  # 印刷機能
    pass

class Shareable:  # 共有機能
    pass

class Document(Saveable, Printable, Shareable):
    # 文書は保存・印刷・共有ができる
    pass
```

❌ 避けるべき使い方

```python
# 悪い例1: 深すぎる階層
class A:
    pass
class B(A):
    pass
class C(B):
    pass
class D(C):
    pass
class E(D):
    pass  # 5階層は深すぎる


# 悪い例2: 無意味な多重継承
class Person(Animal, Vehicle):  # 人は動物で乗り物？
    pass  # 意味不明


# 悪い例3: is-a関係が成り立たない
class Car(Engine):  # 車はエンジンである？
    pass  # 間違い！車はエンジンを「持つ」
```

よくある質問

Q1: どれを使えばいいか迷ったら？

A: まず単一継承から始めましょう

90%のケースは単一継承で十分です。

```python
#迷ったらこれ
class Parent:
  pass

class Child(Parent):
  pass
```

Q2: super()って何？

A: 親クラスのメソッドを呼び出す関数

```python
class Parent:
  def __init__(self, name):
    self.name = name

class Child(Parent):
  def __init__(self, name, age):
    super().__init__(name)  #親の__init__を呼ぶ
    self.age = age
```

Q3: オーバーライドとは？

A: 親クラスのメソッドを子クラスで上書きすること

```python
class Parent:
  def greet(self):
    print("こんにちは")

class Child(Parent):
  def greet(self):  #オーバーライド
    print("Hello!")
```

Q4: 階層継承は何階層まで？

A: 2-3階層まで（最大4階層）

```python
```

```python
# ✅ 良い
class A:
    pass
class B(A):
    pass
class C(B):  # 3階層OK
    pass

# ❌ 悪い
class D(C):
    pass
class E(D):
    pass  # 5階層は深すぎる
```

Q5: 多重継承の注意点は？

A: MROとダイヤモンド問題

```python
# ダイヤモンド継承
class A:
    def method(self):
        print("A")

class B(A):
    def method(self):
        print("B")

class C(A):
    def method(self):
        print("C")

class D(B, C):  # どっちのmethodが呼ばれる？
    pass

# MROで確認
print(D.__mro__)
# D → B → C → A の順番
```

まとめ

🎯 重要ポイント3つ

1️⃣ 継承の種類

種類	構造	推奨度
単一継承	`親 → 子`	★★★★★
階層継承	`親 → 子 → 孫`	★★★☆☆
多重継承	`親A,B → 子`	★★☆☆☆

2️⃣ 選択の基準

```
シンプルに拡張 → 単一継承
段階的に特化 → 階層継承
機能を組み合わせ → 多重継承
```

3️⃣ 黄金律

1. **まず単一継承を考える**
2. **階層は浅く**（2-3階層）
3. **多重継承は慎重に**
4. **迷ったらコンポジション**
5. **is-a関係を確認**

📚 学習のステップ

Week 1: 単一継承をマスター

- [ ] 親クラスを作れる
- [ ] 子クラスで継承できる
- [ ] `super()`を使える
- [ ] メソッドをオーバーライドできる

Week 2: 階層継承を理解

- [ ] 3階層の継承を作れる
- [ ] `super()`の連鎖を理解
- [ ] 適切な階層の深さを判断できる

Week 3: 多重継承に挑戦

- [ ] ミックスインパターンを理解
- [ ] MROを理解
- [ ] 複数の親を初期化できる

🐰 うさうさ店長からのメッセージ

みなさん、お疲れ様でした！

継承は最初は難しく感じるかもしれませんが、一歩ずつ進めば必ず理解できます。

初心者の方へ

まずは**単一継承**だけで十分です。 親から子への単純な拡張を練習しましょう。