Pythonのクラス入門：オブジェクト指向への第一歩

プログラミングの世界、特にPythonを学び始めると、「クラス」という言葉に出会うことがあります。これはオブジェクト指向プログラミング（OOP）と呼ばれる考え方の中心的な要素であり、最初は少し難しく感じるかもしれません。しかし、クラスの概念を理解すると、より整理され、効率的で、現実に即したプログラムを作成できるようになります。
このレポートでは、Pythonのクラスについて、初心者の方にも分かりやすく、基本的な概念から具体的な使い方、そしてそのメリットまでを解説していきます。

1. クラスとは何か？：オブジェクトの設計図

Pythonにおける**クラス（Class）とは、簡単に言えばオブジェクトを作成するための「設計図」や「テンプレート」**のようなものです [1]。家を建てる前に設計図が必要なのと同じように、プログラム内で特定の構造や機能を持つ「モノ」（オブジェクト）を作る際に、まずその設計図としてクラスを定義します [3]。
クラスは、その設計図に基づいて作られるオブジェクトが、どのような**データ（属性）を持ち、どのような振る舞い（メソッド）**をするかを定義します [2]。例えば、「車」というクラスを考えると、その設計図には「色」「最高速度」「メーカー名」といったデータ（属性）や、「走る」「止まる」「曲がる」といった振る舞い（メソッド）が定義されるでしょう [6]。
この「設計図」という考え方は、よくたい焼きの型に例えられます [7]。たい焼きの型（クラス）自体はたい焼きではありませんが、その型を使えば、あんこ味、クリーム味など、様々なたい焼き（オブジェクト）をいくつも作ることができます [10]。クラスも同様に、一度定義すれば、それに基づいて具体的なオブジェクトを効率的に生成するための基盤となります [2]。
重要なのは、クラスはあくまで設計図であり、それ自体が具体的な実体を持つわけではないということです [7]。クラスを利用するには、後述する「インスタンス化」というプロセスを経て、具体的なオブジェクトを作成する必要があります [7]。

2. 基本用語をやさしく解説：オブジェクト、インスタンス、属性、メソッド

クラスを理解する上で、いくつか重要な関連用語があります。ここでは、それぞれの用語を簡単な言葉で解説します。

- オブジェクト **(Object)**: プログラミングにおける「モノ」全般を指す言葉です。クラスという設計図に基づいて作られた具体的な実体もオブジェクトですし、Pythonでは数値や文字列、リストなども内部的にはオブジェクトとして扱われています。オブジェクトは、データ（属性）とそれを操作する手順（メソッド）をひとまとめにしたものです [14]。例えば、実際に道路を走っている一台一台の車がオブジェクトに相当します [6]。

- インスタンス **(Instance)**: クラスという設計図（テンプレート）から実際に作成された具体的なオブジェクトのことです [2]。クラスが「たい焼きの型」なら、その型から作られた一つ一つの「たい焼き」がインスタンスです [7]。クラスが「車の設計図」なら、その設計図に基づいて工場で作られた「特定の赤いスポーツカー」や「白いセダン」がインスタンスになります [6]。インスタンスは、クラスで定義された属性やメソッドを持ちますが、属性の値（例えば車の色）はインスタンスごとに異なる場合があります [17]。しばしば「オブジェクト」と「インスタンス」はほぼ同じ意味で使われますが、特にクラスから作られた実体を指す場合に「インスタンス」という言葉がよく用いられます [9]。
- 属性 **(Attribute /** アトリビュート**)**: オブジェクトが持つデータや情報のことです [1]。クラス定義の中で、変数のように定義されます [17]。例えば、「犬」クラスのインスタンスなら、「名前」「年齢」「犬種」などが属性にあたります [19]。「社員」クラスなら、「社員ID」「名前」「役職」「給与額」などが属性です [17]。これらの属性の値は、各インスタンス（個々の犬や社員）で異なる値を保持することができます [17]。
- メソッド **(Method)**: オブジェクトが行うことができる操作や振る舞いのことです [1]。クラス定義の中で、関数のように定義されます [17]。メソッドは通常、そのオブジェクト自身の属性にアクセスしたり、変更したりするために使われます [17]。「犬」クラスのインスタンスなら、「吠える」「走る」「尻尾を振る」といった動作がメソッドにあたります [14]。「社員」クラスなら、「給与を上げる」「部署を移動する」といった操作がメソッドです [17]。

これらの関係性を寿司に例えると、「寿司はシャリの上にネタを載せたもの」という概念が「クラス」であり、実際にシャリの上にイカやマグロの切り身を載せた個々の寿司が「インスタンス（オブジェクト）」、「ネタの種類」や「シャリの量」が「属性」、「食べる」という行為が「メソッド」と考えることができます [17]。

3. クラスの作り方：設計図を書く

実際にPythonでクラス（設計図）を作成する方法を見ていきましょう。

基本的なクラス定義の構文

Pythonでクラスを定義するには、class キーワードの後に、クラスの名前（通常は大文字で始めるのが慣習）を記述し、コロン : をつけます。クラスの内容（属性やメソッドの定義）は、その次の行からインデント（字下げ）して記述します [5]。

Python

```
class クラス名:
    # クラスの内容（属性やメソッド）をここに記述
    pass # passは何もしない命令。空のクラスを作る際に使う
```

pass は、文法的には何か記述が必要だが、具体的な処理をまだ書かない場合に使用するプレース

ホルダーです。まずは、このように空のクラスを定義することから始められます [21]。Pythonでは、クラス名は大文字で始まるキャメルケース（例: MyClass, Dog, CarModel）にするのが一般的です [22]。

簡単な例：**Dog** クラスを作ってみる

例として、犬を表す Dog クラスを作ってみましょう。まずは最もシンプルな形で定義します。

Python

```
class Dog:
    pass # まずは空のクラスから
```

これだけでは何もできませんが、これがクラス定義の第一歩です。これから属性やメソッドを追加していきます。

クラス内に属性とメソッドを記述する方法

クラスの設計図には、データ（属性）と操作（メソッド）を記述します。

- 属性の定義: オブジェクトが持つべきデータを定義します。各インスタンス（個々の犬）に固有の属性（名前や年齢など）は、後で説明する特別なメソッド __init__ の中で self.属性名 = 値 のように設定するのが一般的です [5]。
- メソッドの定義: オブジェクトができる操作を定義します。クラス内で関数を定義すると、それがメソッドになります。メソッドを定義する際には、最初の引数として self を指定するのがPythonの慣習です [5]。

ここで重要なのが self です。メソッドが呼び出されたとき、Pythonはそのメソッドを呼び出したインスタンス自身を自動的に最初の引数として渡します。この受け皿となるのが self です。メソッド内で self を使うことで、そのインスタンス自身の属性（例: self.name）にアクセスしたり、他のメソッド（例: self.other_method()）を呼び出したりできます。つまり、self は、メソッドとそのメソッドが操作する対象のインスタンスを結びつける橋渡しの役割を果たします。これにより、同じ bark メソッドでも、「ポチ」インスタンスが呼び出せばポチの名前を使って吠え、「コロ」インスタンスが呼び出せばコロの名前を使って吠える、といったオブジェクトごとの振る舞いが可能になります。
__init__ を使う前の段階として、簡単な属性とメソッドを持つ Dog クラスの例を示します。（属性の定義方法は後ほど __init__ で洗練させます）

Python

```
class Dog:
    # 属性（仮置き。通常は __init__ で設定）
    # name = "Pochi" # この書き方だと、全ての犬が"Pochi"になってしまう
```

```
    # メソッド
    def bark(self, message): # self の他に引数を取ることも可能
        # print(f"{self.name} says: {message}") # name属性がないとエラーになる
        print(f"Someone says: {message}") # self.name を使わない例

    def wag_tail(self):
        # print(f"{self.name} wags tail.") # name属性がないとエラーになる
        print("Tail wagging!") # self.name を使わない例
```

この例では、まだインスタンスごとの名前 (name) をうまく扱えていませんが、クラス内に def を使ってメソッド (bark, wag_tail) を定義する基本的な形を示しています。メソッド定義の際には self を第一引数に取ることを忘れないでください。

4. 設計図から実体を作る：インスタンス化

クラスという設計図を定義しただけでは、プログラム内で実際に使うことはできません [7]。設計図をもとに、具体的な「モノ」、つまりインスタンス（オブジェクト）を作成する必要があります。このプロセスをインスタンス化 **(Instantiation)** と呼びます [5]。
インスタンス化を行うには、クラス名の後ろに括弧 () をつけて呼び出します。これは関数を呼び出すのと似た形式です [13]。これにより、クラス定義に基づいた新しいオブジェクトがメモリ上に生成され、プログラムで扱えるようになります [4]。

コード例：**Dog** クラスからインスタンスを生成する

それでは、先ほど定義した（あるいは後述の __init__ を持つ）Dog クラスから、実際の犬のインスタンスを作成してみましょう。

Python

```
# Section 6 で定義する __init__ を持つ Dog クラスを仮定
class Dog:
    def __init__(self, name): # インスタンス化時に名前を受け取る
        self.name = name      # 受け取った名前をインスタンスの属性として保存
        print(f"Dog instance named '{self.name}' created!")

    def bark(self):
        print(f"{self.name} says Woof!")

# インスタンス化：クラス名() で呼び出す
my_dog1 = Dog("Pochi") # Dogクラスから "Pochi" という名前のインスタンスを作成
my_dog2 = Dog("Koro")  # Dogクラスから "Koro" という名前のインスタンスを作成
```

```
# 作成されたインスタンスを確認（メモリ上の別々のオブジェクトであることがわかる）
print(my_dog1)
print(my_dog2)
```

このコードを実行すると、Dog("Pochi") と Dog("Koro") の呼び出しによって、それぞれ my_dog1 と my_dog2 という名前の変数に、Dog クラスのインスタンスが代入されます。print 文の出力を見ると、my_dog1 と my_dog2 はメモリ上の異なる場所に存在する、独立したオブジェクトであることが確認できます。このように、一つのクラスから複数の、しかしそれぞれが独立したインスタンスを作り出すことができるのがクラスの強力な点です。

5. 作成したオブジェクトを操作する

クラスからインスタンスを作成したら、そのインスタンスの属性にアクセスしたり、メソッドを呼び出したりして操作します。

属性へのアクセス方法（値の参照・変更）

インスタンスが持つ属性（データ）にアクセスするには、ドット . 演算子を使います。インスタンス名.属性名 のように記述します [17]。これにより、属性の値を読み取ったり、新しい値を代入して変更したりできます。

Python

```
# my_dog1 は Dog("Pochi") で作成済みとする
print(my_dog1.name) # my_dog1 インスタンスの name 属性を参照 -> Pochi

# 属性の値を変更する
my_dog1.name = "Hachi"
print(my_dog1.name) # 変更後の name 属性を参照 -> Hachi
```

このように、ドット演算子を使って、オブジェクトの状態（属性値）を確認したり、更新したりすることができます。

メソッドの呼び出し方

インスタンスのメソッド（操作）を呼び出す際も、属性へのアクセスと同様にドット . 演算子を使用します。インスタンス名.メソッド名() のように記述します [14]。メソッド名の後には、引数がない場合でも必ず括弧 () をつける必要があります。

Python

```
# my_dog1 は Dog("Hachi") で作成済みとする
my_dog1.bark() # my_dog1 インスタンスの bark メソッドを呼び出す -> Hachi says Woof!

# my_dog2 (名前は "Koro") のメソッド呼び出し
# my_dog2 は Dog("Koro") で作成済みとする
my_dog2.bark() # my_dog2 インスタンスの bark メソッドを呼び出す -> Koro says Woof!
```

この例からわかるように、同じ bark メソッドを呼び出しても、my_dog1 と my_dog2 では、それぞれの name 属性の値に基づいて異なる出力（"Hachi says Woof!" と "Koro says Woof!"）が得られます。これは、メソッドが self を通じて、呼び出されたインスタンス自身の属性にアクセスしているためです。

コード例：**Dog** インスタンスの属性アクセスとメソッド実行

インスタンスの作成から属性アクセス、メソッド呼び出しまでの一連の流れをまとめると、以下のようになります。

Python

```
# 再掲: __init__ を持つ Dog クラス
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
        print(f"Dog instance named '{self.name}', age {self.age} created!")

    def bark(self):
        print(f"{self.name} says Woof!")

    def celebrate_birthday(self):
        self.age += 1
        print(f"Happy Birthday {self.name}! You are now {self.age} years old.")

# 1. インスタンス化
pochi = Dog("Pochi", 3)
koro = Dog("Koro", 1)

# 2. 属性へのアクセス (参照)
print(f"{pochi.name} is {pochi.age} years old.")
print(f"{koro.name} is {koro.age} years old.")
```

```
# 3. メソッドの呼び出し
pochi.bark()
koro.bark()

# 4. メソッド呼び出しによる属性の変更
pochi.celebrate_birthday()

# 5. 変更された属性の確認
print(f"{pochi.name} is now {pochi.age} years old.")
```

この例では、pochi と koro という二つの独立した Dog インスタンスを作成し、それぞれの名前や年齢を参照したり、bark メソッドを呼び出したりしています。さらに、celebrate_birthday メソッドを呼び出すことで、pochi の age 属性が内部で変更され、その結果を確認しています。このように、インスタンスごとに状態（属性）を持ち、メソッドによってその状態が変化したり、状態に基づいた振る舞いをしたりするのが、オブジェクト指向プログラミングの基本的な考え方です。

6. 初期設定の自動化：__init__ メソッド（コンストラクタ）

クラスからインスタンスを作成する際に、多くの場合、そのインスタンスが最初から持っておくべき属性（名前、年齢など）を設定したいと考えます。例えば、Dog インスタンスを作るなら、作成と同時に名前と年齢を設定できると便利です。これを実現するのが、__init__ という特別なメソッドです。

__init__ の役割：インスタンス生成時の初期化

__init__ メソッドは、クラスからインスタンスが生成される直後に自動的に呼び出される特別なメソッドです [17]。これは「コンストラクタ (Constructor)」とも呼ばれ、その主な役割は、生成されたインスタンスの**初期設定（初期化）**を行うことです [5]。具体的には、インスタンスが持つべき属性（インスタンス変数）に初期値を設定します [5]。
__init__ の名前の前後に付いている二つのアンダースコア __ は、これがPythonにおいて特別な意味を持つメソッドであることを示しています。

self 引数の重要性

__init__ メソッドを定義する際、最初の引数は必ず self にします [5]。この self は、まさに生成されつつあるインスタンス自身を指します。__init__ メソッドの中で self.属性名 = 値 という形でコードを書くと、その新しく作られたインスタンスに属性が追加され、値が設定されます [5]。
インスタンス化を行う際（例: Dog("Pochi", 3)）、クラス名に渡された引数（この例では "Pochi" と 3）は、__init__ メソッドの self に続く引数として受け取られます。

Python

```
class Dog:
    # コンストラクタ (__init__ メソッド)
    def __init__(self, name, age): # self の次に name と age を受け取る
        print(f"Initializing Dog instance...")
        # 受け取った引数を使って、インスタンスの属性を設定
        self.name = name  # self (インスタンス自身) の name 属性に name 引数の値を設定
        self.age = age    # self (インスタンス自身) の age 属性に age 引数の値を設定
        print(f"Dog named {self.name}, age {self.age} created.")

    def bark(self):
        print(f"{self.name} says Woof!")

# インスタンス化：__init__ が自動で呼び出される
# Dog("Pochi", 3) は、内部的に __init__(self, "Pochi", 3) を呼び出す
dog1 = Dog("Pochi", 3)
dog2 = Dog("Koro", 1) # こちらは __init__(self, "Koro", 1) を呼び出す

# __init__ で初期化された属性にアクセスできる
print(f"{dog1.name} is {dog1.age} years old.") # -> Pochi is 3 years old.
print(f"{dog2.name} is {dog2.age} years old.") # -> Koro is 1 years old.
```

__init__ を使うメリット

__init__ を使う最大の利点は、インスタンスが必要なデータを確実に持って生成されることを保証できる点です。もし __init__ を使わずに、インスタンス作成後に手動で属性を設定しようとすると（例: dog1 = Dog(), dog1.name = "Pochi", dog1.age = 3）、属性を設定し忘れたり、タイプミスをしたりする可能性があります [24]。
__init__ を定義しておけば、インスタンス化の際に必要な情報（引数）を渡さないとエラーになるため、不完全な状態のオブジェクトが作られるのを防ぐことができます。これにより、オブジェクトが常に予測可能で一貫した状態から始まるため、コード全体の信頼性と保守性が向上します [17]。__init__ は、いわばオブジェクトが正しく「生まれる」ための必須手続きを定義する役割を担っているのです。

7. クラスを利用するメリット

なぜわざわざクラスを使うのでしょうか？クラスには、プログラミングを行う上で多くの利点があります。

- コードの整理整頓と可読性向上: クラスは、関連性の高いデータ（属性）とそれを操作する処理（メソッド）を一つの「まとまり」としてカプセル化します [1]。例えば、「犬」に関連するデータ（名前、年齢）と操作（吠える、尻尾を振る）が Dog クラスの中に集約されるため、コードのどこに何が書かれているかが分かりやすくなります。これにより、プログラム全体の構造が明確になり、他の人が読んだり、後で自分が見返したりする際の可読性が大幅に向上します [3]。手続き

型プログラミングのように関連する変数や関数がバラバラに存在する場合と比較して、管理が容易になります [15]。

- コードの再利用性の向上: 一度クラス（設計図）を定義してしまえば、そのクラスから必要なだけインスタンス（実体）を簡単に生成して再利用できます [2]。例えば、たくさんの犬をプログラムで扱いたい場合でも、Dog クラスを一つ定義しておけば、Dog("ポチ"), Dog("コロ"), Dog("ハチ") のように、インスタンス化するだけで済みます。同じような機能を持つコードを何度も繰り返し書く必要がなくなり、効率的な開発が可能になります [7]。これは、プログラミングにおける重要な原則であるDRY（Don't Repeat Yourself - 同じことを繰り返すな）の実践にも繋がります [10]。
- 現実世界の概念をモデル化しやすい: クラスは、私たちが普段認識している現実世界の「モノ」（犬、車、社員、商品など）や「概念」（座標、注文、予約など）を、プログラムの中で自然に表現（モデル化）するための強力な手段です [4]。オブジェクト指向プログラミング（OOP）の基本的な考え方は、このように現実世界のエンティティをソフトウェア内のオブジェクトとして表現することにあり、クラスはそのための主要なツールとなります [1]。これにより、問題領域の構造をコードの構造に反映させやすくなり、より直感的で理解しやすいプログラム設計が可能になります [9]。

これらのメリットは互いに関連し合っています。コードが整理され（整理整頓）、再利用可能になり（再利用性）、現実の概念をうまく反映できる（モデル化）ことで、結果として保守性（修正や機能追加のしやすさ）が高く、拡張性（将来的な変更への対応力）に優れた、より質の高いソフトウェアを開発することが可能になります [15]。クラスは、単なる機能の集まりではなく、より良いソフトウェア設計を実現するための基礎となるのです。

8. まとめ：クラス理解への第一歩

このレポートでは、Pythonのクラスに関する基本的な概念と使い方を解説しました。最後に、重要なポイントを振り返りましょう。

- クラスは設計図: オブジェクトを作るためのテンプレートです [1]。
- インスタンスは実体: クラスから作られた具体的なオブジェクトです [2]。
- 属性はデータ: オブジェクトが持つ情報（変数）です [14]。
- メソッドは操作: オブジェクトができること（関数）です [14]。
- インスタンス化: クラスからインスタンスを作成するプロセスです (クラス名()) [5]。
- **__init__** はコンストラクタ: インスタンス生成時に自動で呼ばれ、初期設定を行います [17]。
- **self** はインスタンス自身: メソッド内でインスタンス自身の属性やメソッドにアクセスするために使います [5]。
- クラスのメリット: コードの整理、再利用性の向上、現実世界のモデル化に役立ちます [3]。

クラスは、オブジェクト指向プログラミングの基礎であり、Pythonプログラミングにおいて非常に重要な概念です [15]。今回学んだ基本を土台として、ぜひご自身で簡単なクラスを作成し、インスタンス化して操作する練習をしてみてください。

クラスの基本を理解したら、次のステップとして継承 **(Inheritance)** という概念を学ぶことをお勧めします [14]。継承を使うと、既存のクラスの機能を引き継いだ新しいクラスを作成でき、コードの再利用性

をさらに高めることができます。また、インスタンス間で共有されるクラス変数[17]や、インスタンス化せずにクラスから直接呼び出せるクラスメソッド、スタティックメソッド[25]など、さらに進んだクラスの機能もあります。
クラスを使いこなせるようになると、より複雑で大規模なプログラムも、構造的に、そして効率的に開発できるようになるでしょう。

引用文献

1. Pythonのオブジェクト指向プログラミングを完全理解 - Qiita, 4月 24, 2025にアクセス、https://qiita.com/kaitolucifer/items/926ed9bc08426ad8e835
2. Pythonのクラスについて #初心者 - Qiita, 4月 24, 2025にアクセス、https://qiita.com/eric50905/items/3030cfa03f849fec04dd
3. Pythonの応用文法【クラス】| サクッと始めるプログラミング入門【Python】- Zenn, 4月 24, 2025にアクセス、https://zenn.dev/umi_mori/books/python-programming/viewer/python-advanced-classes
4. [Python入門]クラスの基礎知識:Python入門(1/2 ページ) - @IT - ITmedia, 4月 24, 2025にアクセス、https://atmarkit.itmedia.co.jp/ait/articles/1907/26/news020.html
5. pythonのclass徹底解説:オブジェクト指向プログラミングへの第一歩 | データと統計学, 4月 24, 2025にアクセス、https://df-learning.com/python_class/
6. Python初心者必見！10分で理解するクラスとオブジェクト指向プログラミング - Code Begin, 4月 24, 2025にアクセス、https://code-begins.com/archives/609
7. Python | class(クラス)の基本・使い方と定義方法 - dot blog, 4月 24, 2025にアクセス、https://dot-blog.jp/news/python-class/
8. 【Python超入門コース】13.クラス | クラスとは、「データ」と「処理」をまとめたもの【プログラミング初心者向け入門講座】- キノブログ, 4月 24, 2025にアクセス、https://kino-code.com/course-python13-class/
9. 「オブジェクト指向言語で楽々プログラミング」, 4月 24, 2025にアクセス、https://shintani.fpark.tmu.ac.jp/classes/open_univ_old/OpenUniversityPythonClass.pdf
10. Pythonのclassとは？使い方の基本を現役エンジニアが徹底解説【初心者向け】, 4月 24, 2025にアクセス、https://magazine.techacademy.jp/magazine/22376
11. Pythonのクラスメソッドの使い方について現役エンジニアが解説【初心者向け】, 4月 24, 2025にアクセス、https://magazine.techacademy.jp/magazine/27996
12. 【Kotlin超入門コース】13.クラス | クラスとは、「データ」と「処理」をまとめたもの【プログラミング初心者向け入門講座】- キノブログ, 4月 24, 2025にアクセス、https://kino-code.com/course-kotlin13-class/
13. 【Python】クラスという設計図を使ってコードをわかりやすくしよう - note, 4月 24, 2025にアクセス、https://note.com/keyma4note/n/nbb0cff6374ef
14. イラストでわかる！Python のオブジェクト指向 | キカガクブログ, 4月 24, 2025にアクセス、https://www.kikagaku.co.jp/kikagaku-blog/python-object-orientation/
15. 【初心者向け】クラスとは何か？ - AI Academy Media, 4月 24, 2025にアクセス、https://aiacademy.jp/media/?p=131
16. 【Python】オブジェクト指向プログラミングの基礎と実装方法を解説 - Udemy メディア, 4月 24, 2025にアクセス、

https://udemy.benesse.co.jp/development/python-work/python-object-oriented.html
17. Pythonにおけるクラスとインスタンス【初心者向け解説記事】, 4月 24, 2025にアクセス、https://workteria.forward-soft.co.jp/blog/detail/10982
18. Pythonのクラス学習。オブジェクト指向と継承 - Qiita, 4月 24, 2025にアクセス、https://qiita.com/The_Boys/items/7c4a04c1412c79c220ef
19. そもそもインスタンスとは - Pythonの学習帳, 4月 24, 2025にアクセス、https://beginner-engineers.com/instance/
20. 初心者でもわかるPython「クラス」入門 - エンベーダー, 4月 24, 2025にアクセス、https://envader.plus/article/16
21. 【python超入門！】クラスやメソッドについて解説します。| 侍エンジニアブログ, 4月 24, 2025にアクセス、https://www.sejuku.net/blog/72161
22. Pythonでclassを定義するには？どんな場合に必要？ - TECH PLAY Magazine, 4月 24, 2025にアクセス、https://techplay.jp/column/492
23. python classの使い方！基礎から応用まで解説 - AI研究所, 4月 24, 2025にアクセス、https://ai-kenkyujo.com/programming/language/python/class/
24. Pythonのクラス学習。挫折からリベンジする話。- Qiita, 4月 24, 2025にアクセス、https://qiita.com/The_Boys/items/1ceef44d7b2c5ce5dacb
25. 【徹底解説】Pythonのクラスの基本からクラス継承やクラス変数などまでわかりやすく - YouTube, 4月 24, 2025にアクセス、https://m.youtube.com/watch?v=Q5eY2I46qmA&pp=ygUfI-eiuueOh-eahOODI-ODreOCsOODqeODn-ODs-OCsA%3D%3D
26. 【Python】オブジェクト指向を理解するための超重要ワードまとめ - DjangoBrothers, 4月 24, 2025にアクセス、https://djangobrothers.com/blogs/basic_knowledge_of_python/