Python 用語

「オブジェクト指向」

基本的に、「オブジェクト」で構成されているという考え方。

Python では、クラスや関数も全て「オブジェクト」として扱われる。

「クラス」

設計図(機能を持たせる)

ただコードが書いてあるだけ。

※クラスを定義するときは、データとデータの振る舞い（メソット）を定義する必要がある。

クラス変数：クラス内の変数。インスタンスごとで共有可能。クラス定義文の直下に代入。

|  |
| --- |
| **class GirlFriend(object):**  **# クラス変数**  **max\_intimacy = 100** |

「インスタンス」

クラスから生成されたオブジェクトを「インスタンス」という。

|  |
| --- |
| “物体の設計図がクラスであれば、その設計図をもとに作られた製品がインスタンス”というイメージ |

インスタンス変数：関数定義文の直下で代入された**self**の属性に代入

|  |
| --- |
| **class GirlFriend(object):**  **def \_\_init\_\_(self, new\_name=''):**  **# インスタンス変数**  **self.name = new\_name** |

「メソッド」

クラス内の関数

「コンストラクタ」

クラスの中にある、’\_\_init\_\_’というなメソット。クラスを呼んだ時に最初に呼ばれる関数。

インスタンスを作成するのに、最も重要な処理を含むもの。

‘self’は、自身をを指す。

例（messageを表示するインスタンスを生成するクラス）

|  |
| --- |
| **class MyClass():**  **def \_\_init\_\_(self, message):#コンストラクタ**  **self.value = message**  **※ここまでがclass！！**  **myinstance = MyClass("Hello!")**  **print(myinstance.value)**  **実行結果**  **"Hello!"** |

**‘self.value = message’**

**インスタンスに‘message’を追加したいので、‘value’という属性を追加し、第二引数を代入。**

**‘myinstance = MyClass("Hello!")’**

**’myinstance’というインスタンスが生成された。これはclass全体を示している。**

**"Hello!" がmessageに入り、‘self.value = message’で、messageのインスタンスが生成。**

**つまり、myinstance.value をプリントすると"Hello!"が表示される。**

「属性」

ドットに続く全ての名前に使われる。

（例）z.real realはオブジェクトzの属性。※整数を生み出すなど、様々な属性がある。

オブジェクトがどのような属性を持っているかを調べるためには、以下のようなコードで調べると表示される。

|  |
| --- |
| **ls = dir(オブジェクト)** |

クラスについてもう少し詳しく見ていく。

例１

|  |
| --- |
| **class Spam:**  **※valはclass変数**  **spam = Spam()**  **１.class Spamのインスタンス‘spam’を生成。**  **※spamは、class全体を指している。**  **spam.ham()**   1. **インスタンス‘spam’がhamメソッドを呼ぶ。** 2. **hamメソッドがeggメソッドを呼ぶ。**  * **msg = 'call method'**   **eggメソッドで、‘msg’と‘self.val’が出力される。**  ※**self.valとなっているのは、valをインスタンスに追加したいため。**  **val = 100**  **def ham(self):**  **self.egg('call method')**  **def egg(self,msg):**  **print("{0}".format(msg))**  **print("{0}".format(self.val))**  **spam = Spam()**  **spam.ham()**  **出力結果**  **call method**  **100** |

例２（コンストラクタ）

|  |
| --- |
| **class Spam:**  **spam = Spam(5,10)**  **1.ham＝5，egg＝10のインスタンスを生成。**  **※spamは、class全体を指している。**  **2.コンストラクタ(\_\_init\_\_)でデータの初期設定**  **※ham，eggのインスタント変数生成。**  **spam.output()**  **1. spamがoutput()メソッドを呼ぶ。**  **2.hamとeggのインスタンス変数の和をsumとする。**  **3. sumを出力する。**  **def \_\_init\_\_(self,ham,egg):**  **self.ham = ham**  **self.egg = egg**  **def output(self):**  **sum = self.ham + self.egg**  **print("{0}".format(sum))**  **spam = Spam(5,10)**  **spam.output()**  **出力結果**  **15** |

例（継承）

２つのクラスが親子関係を持ち、子クラスは親クラスの性質を受け継ぐ機能。

※ここでは、**Base**が親クラス。**Derived**が子クラス。

|  |
| --- |
| **class Base:**  **derived = Derived()**  **1.クラスDerivedのインスタンスderived を生成。**  **derived.basevalue**   1. **インスタンスderivedが、親クラスのbasevalueを呼ぶ。** 2. **printにより、base が出力。**   **derived.ham()**  **1.インスタンスderivedが、親クラスのメソッドham()を呼ぶ。**  **2. printにより、ham が出力。**  **ちなみに、**  **derived.spam()とすると…**   1. **クラスDerived()のメソッドspamが呼ばれる。** 2. **printにより、Derived.spam()が出力。** 3. **self.ham()により、親クラスのメソッドhamfが呼ばれる。** 4. **printにより、ham が出力。**   **basevalue = "base"**  **def spam(self):**  **print("Base.spam()")**  **def ham(self):**  **print("ham")**  **class Derived(Base):**  **def spam(self):**  **print ("Derived.spam()")**  **self.ham()**  **derived = Derived()**  **print("{0}".format(derived.basevalue))**  **derived.ham()**  **出力結果**  **base**  **ham** |

「Python のクラスにおける\_\_call\_\_メソッドの使い方」

一度生成されたインスタンスを呼び出し、関数のように使える。

|  |
| --- |
| **class A:**  **a = A(1)**  **1.クラスAのコンストラクタメソッドを呼ぶ。**  **※a = 1**  **2.Aをインスタンスとして定義する。**  **printによって、A init が出力。**  **a(2)**  **1.クラスAのコンストラクタは、すでに定義されているので、\_\_call\_\_メソッドを呼ぶ。**  **※b = 2**  **2.printによって、A init が出力。**  **3.先にコンストラクタメソッドで定義されたaを呼び、b + self.a の計算結果を、printによって、出力。**  **b = B(1,3)**   1. **クラスBのコンストラクタを呼ぶ。**   **※a = 1, c = 3**  **2.super().\_\_init\_\_(a)によって、aは親クラスAのコンストラクタで実行される。**  **※aをインスタンスとして定義。A init を出力。**  **3. cをインスタンスとして定義。B init を出力。**  **b(4)**   1. **クラスBのコンストラクタは、すでに定義されているので、\_\_call\_\_メソッドを呼ぶ。**   **※d = 4**  **2. B callを出力**  **3.先にコンストラクタメソッドで定義されたa,cを呼び、self.a + self.c + dの計算結果を出力。**  **def \_\_init\_\_(self, a):**  **self.a = a**  **print("A init")**  **def \_\_call\_\_(self, b):**  **print("A call")**  **print(b + self.a)**  **class B(A):**  **def \_\_init\_\_(self, a, c):**  **super().\_\_init\_\_(a)**  **self.c = c**  **print("B init")**  **def \_\_call\_\_(self, d):**  **print("B call")**  **print(self.a + self.c + d)**  **出力結果**  **>>a = A(1)**  **A init**  **>>** **a(2)**  **A call**  **3**  **>>** **b = B(1,3)**  **A init**  **B init**  **>>** **b(4)**  **B call**  **8** |

参考URL：

<https://www.sejuku.net/blog/28182> （オブジェクト指向、インスタンス、クラス、コンストラクタ）

<https://docs.python.jp/3/tutorial/classes.html> (属性)

<https://qiita.com/Usek/items/a206b8e49c02f756d636> （クラスの作成の仕方）

<http://yoshi-python.blogspot.jp/2009/10/blog-post_09.html> （オブジェクトから属性を取得する）

<http://nihaoshijie.hatenadiary.jp/entry/2018/01/15/225346> （インスタンス変数、クラス変数）

<https://qiita.com/kyo-bad/items/439d8cc3a0424c45214a> (\_\_call\_\_とはどのような関数なのか)