

# プログラミング初級 (Python)

---

オブジェクト指向

早稲田大学グローバルエデュケーションセンター

# プログラミングのパラダイム

---

単なる技術ではなく、思考法である。

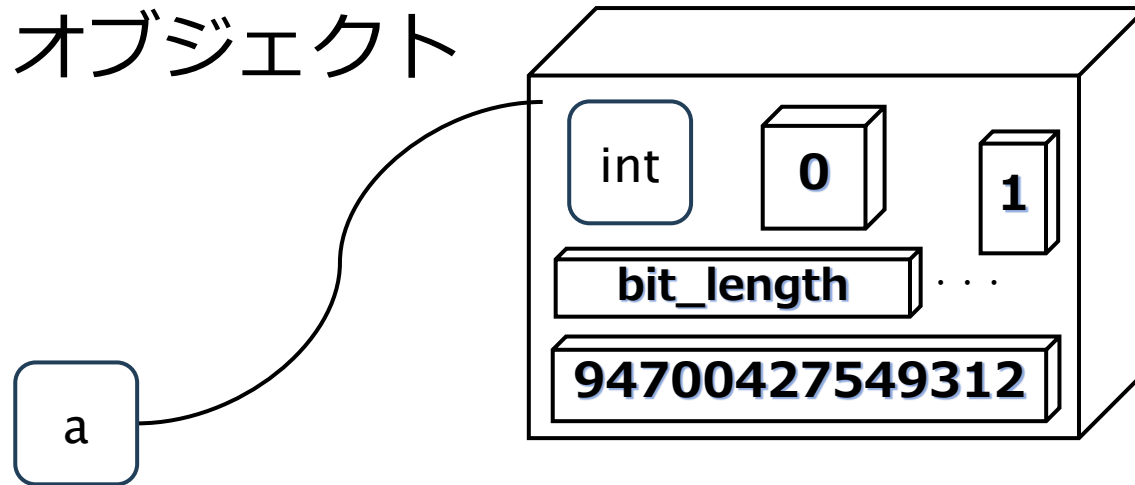
— アラン・ケイ

---

# オブジェクトとは？

- Pythonに含まれるものは数値から関数に至るまで全てが**オブジェクト**。
- しかし、Pythonではこの点をあまり意識せずともプログラミング可能。
- 例えば、`a = 0` とかけば、Pythonが自動的に値0の整数型のオブジェクトを生成し、さらにそのオブジェクトへの参照も自動的に生成する。

オブジェクト



For now, leave  
the difficult  
things to me!

- オブジェクトは平たく捉えれば、**関連するデータの塊、チャンク**。
- オブジェクトは技術的には**関連する変数や値、関数のデータ構造**。
- オブジェクトのうち、変数や値は**属性**、関数は**メソッド**という。

# オブジェクト指向とは？

- ・我々人間は、無意識にかこの世の**物事を状態や動作の違いによって知覚**することが多い。物事の例：無機物、生き物、概念、など。
- ・コンピュータプログラムでは、物事のことを**オブジェクト**と呼び、**状態に相当するものを属性、動作に相当するものをメソッド**と呼ぶ。
- ・見方を変えてみると、**物事には固有の状態や動作**がある。
  - ・状態の例：見た目、大きさ、重さ、情報、など。
  - ・動作の例：喋る、飛ぶ、食べる、変わる、など。
- ・であれば、本来まとまっているはずの**状態や動作を点在化させず**に、関連するデータを物事としての単位または**オブジェクト単位**で**プログラムを書いたり、まとめたり**することで、**コンピュータプログラムを人間の知覚方法に近い表現にして分かりやすくしていこうという考え方が**生まれた。
- ・このようなプログラミングの**思考法、概念的枠組み、パラダイム**のことを**オブジェクト指向**という。

# 手続き型プログラミングとの比較

- オブジェクト指向ではなく、**作業の手順**のみを指向してきたこれまでのプログラミング手法は**手続き型プログラミング**と呼ばれる。
- 手続き型プログラミングでは、データ構造はタプル、リスト、辞書、集合などのレベルで適宜まとめられるが、その他のまとめ方ではなく、**変数や値、関数がプログラムでしばしば点在化する**。
- オブジェクト指向プログラミングでは、**関連性の高い変数や値、そして関数**が属性またはメソッドとしてオブジェクトにまとめられるため、それらが**点在化することはない**。
- **オブジェクトの単位で命令をまとめるという制約**により、**想定外の処理が起こり得るリスクを軽減**できる。想定外の処理の例：ブーリアンを大文字化？→大文字もしくは小文字に変わるのは文字列が持てば良い動作といえる。電子レンジが空を飛ぶ？→空を飛ぶのは鳥類や飛行物体が持てば良い動作といえる。
- 現実世界における物事も瞬時に劇的な変化は起こらないものの、それによる悪影響もない。**オブジェクト指向は現実のルールに則している**。

# クラスの定義

- Pythonにおいて標準で利用可能なデータ型はデフォルトのオブジェクトとして定義されているため、プログラマが定義する必然性はない。
- 変数や関数と同様に、**独自のデータ構造を作成する場合は**プログラマが新しいオブジェクトとして独自に**クラスを定義**する必要がある。クラスの定義には、**class文**を用いる。

オブジェクトを知覚しやすいよう分類しているイメージを持つと良い。

構文

```
class object_name:  
    """docstring"""  
    変数  
    関数
```

- 以前の構文同様、**ブロックを定めるためのインデント**の記述が必要。
- docstring、変数、関数が揃わなければクラスではないという訳ではない。どの要素も必要に応じて省略してよい。
- object\_nameはクラスの名前を表す。**命名規則は変数や関数と同じ**。
- **クラスはサブルーチンの一種であり、あくまで定義であるので、必要な際は関数と同様に呼び出して利用**する。

# クラスの定義：属性とメソッド

```
1 >>> class Bear:
...     pass
...
2 >>> dir(Bear)
['__class__', '__delattr__', '__dict__',
 '__dir__', '__doc__', '__eq__',
 '__format__', '__ge__',
 '__getattr__', '__getstate__',
 '__gt__', '__hash__', '__init__',
 '__init_subclass__', '__le__', '__lt__',
 '__module__', '__ne__', '__new__',
 '__reduce__', '__reduce_ex__', '__repr__',
 '__setattr__', '__sizeof__', '__str__',
 '__subclasshook__', '__weakref__']
3 >>> Bear.name = 'Teddy'
4 >>> dir(Bear)
```

クラス名は大文字で始めることが多い。

クラスを定義した後に属性を作成する方法。

```
['__class__', '__delattr__', '__dict__',
 '__dir__', '__doc__', '__eq__',
 '__format__', '__ge__',
 '__getattr__', '__getstate__',
 '__gt__', '__hash__', '__init__',
 '__init_subclass__', '__le__', '__lt__',
 '__module__', '__ne__', '__new__',
 '__reduce__', '__reduce_ex__', '__repr__',
```

```
'__setattr__', '__sizeof__', '__str__',
 '__subclasshook__', '__weakref__', 'name']
5 >>> Bear.name
'Teddy'
6 >>> class Bear:
...     name = 'Teddy'
...
7 >>> Bear.name
'Teddy'
8 >>> class Bear:
...     name = 'Teddy'
...     def eat(food):
...         print(f"{food}. Yum!")
...
9 >>> Bear.name; Bear.eat('Fish')
'Teddy'
Fish. Yum!
10 >>> False.__and__(True)
False
11 >>> (3).__bool__(); (3.14).__floor__()
True
3
12 >>> 'hoge'.upper()
'HOGE'
```

属性へのアクセス。

オブジェクトの要素にアクセスするのに変数は必須ではない。

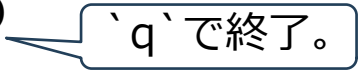
# クラスの定義：docstring、変数のスコープ

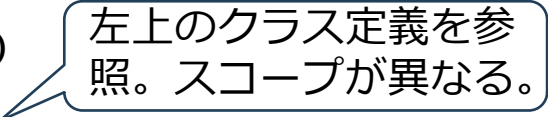
```
13 >>> class Bear:
...     """
...     Class representing a bear.
...     """
...     name = 'Teddy'
...     def eat(food):
...         print(f"{name} is eating {food}.
Yum!")
...
14 >>> help(Bear)

Help on class Bear in module __main__:

class Bear(builtins.object)
|   Class representing a bear.
|
|   Methods defined here:
|
|   eat(food)
|
|   -----
|   -----
```

```
|   Data descriptors defined here:
|
|   __dict__
|       dictionary for instance variables
(if defined)
|
|   __weakref__
|       list of weak references to the
object (if defined)
|
|   -----
|   -----
|   Data and other attributes defined here:
|
|   name = 'Teddy'
(END)
15 >>> print(Bear.__doc__)
    Class representing a bear.
16 >>> Bear.eat('Fish')
    NameError: name 'name' is not defined
```

 `q`で終了。

 左上のクラス定義を参照。スコープが異なる。



# インスタンスの生成

```
17 >>> class Bear:
...     """
...     Class representing a bear.
...     """
...     name = 'Teddy'
...     def eat(food):
...         print(f"{food}. Yum!")
...
18 >>> a_bear = Bear
19 >>> a_bear.name
'Teddy'
20 >>> a_bear.eat('Fish')
Fish. Yum!
21 >>> a_bear1 = Bear
22 >>> a_bear1.name
'Teddy'
23 >>> a_bear1.name = 'Pooh'
24 >>> a_bear1.name
'Pooh'
25 >>> a_bear.name
'Pooh'
```

他のインスタンスからアクセスした場合でも値が変わっている。`name` はクラス変数といってインスタンス間で共有される。

```
26 >>> class Bear:
...     """
...     Class representing a bear.
...     """
...     def f(instance, name):
...         instance.name = name
...     def eat(instance, food):
...         print(f"{instance.name} is
eating {food}. Yum!")
...
27 >>> a_bear = Bear()
28 >>> a_bear.f('Teddy')
29 >>> a_bear.name
'Teddy'
30 >>> a_bear.eat('fish')
Teddy is eating fish. Yum!
31 >>> a_bear1 = Bear()
32 >>> a_bear1.f('Pooh')
33 >>> a_bear1.name
'Pooh'
34 >>> a_bear1.eat('honey')
Pooh is eating honey. Yum!
```

我輩は熊である。名前はまだない。

インスタンスごとに異なる情報を持たせられる。名前は`Teddy`。

第1引数のインスタンス変数は自動的に取得されるので第2引数のみ入力する。

インスタンスごとに異なる情報を持たせられる。名前は`Pooh`。

# コンストラクタとself

```
35 >>> class Bear:
...     """
...     Class representing a bear.
...     """
...     def __init__(instance, name):
...         instance.name = name
...     def eat(instance, food):
...         print(f"{instance.name} is
eating {food}. Yum!")
...
36 >>> a_bear = Bear('Teddy')
37 >>> a_bear.name
'Teddy'
38 >>> a_bear.eat('fish')
Teddy is eating fish. Yum!
39 >>> a_bear1 = Bear('Pooh')
40 >>> a_bear1.name
'Pooh'
41 >>> a_bear1.eat('honey')
Pooh is eating honey. Yum!
```

コンストラクタ。

第1引数のインスタンス変数は自動的に取得されるので手動で指定するのは第2引数のみ。

クラスにあるコンストラクタが引数を持つ場合はクラスを呼び出す際に( )の中で引数を渡しておく。

第1引数のインスタンス変数は自動的に取得されるので第2引数のみ入力する。

```
42 >>> class Bear:
...     """
...     Class representing a bear.
...     """
...     def __init__(self, name):
...         self.name = name
...     def eat(self, food):
...         print(f"{self.name} is eating
{food}. Yum!")
...
43 >>> a_bear = Bear('Teddy')
44 >>> a_bear.name
'Teddy'
45 >>> a_bear.eat('fish')
Teddy is eating fish. Yum!
46 >>> a_bear1 = Bear('Pooh')
47 >>> a_bear1.name
'Pooh'
48 >>> a_bear1.eat('honey')
Pooh is eating honey. Yum!
```

インスタンスを表す引数名は`self`とすることが慣例となっているが予約語ではない。

熊のインスタンスが生成された段階で既に名前を持っている。

# オブジェクト指向プログラミングの実践

```
1 class Bear:
2     def __init__(self, name):
3         self.name = name
4         self.energy = 100
5
6     def eat(self, food):
7         print(f"{self.name} is eating
8 {food}. Yum!")
9         self.energy += 10
10
11    def sleep(self, hours):
12        print(f"{self.name} is sleeping for
13 {hours} hours. Zzz...")
14        self.energy += hours * 5
15
16    def get_energy(self):
17        return self.energy
18
19 # Create bear objects
20 teddy = Bear("Teddy")
21 pooh = Bear("Pooh")
```

```
21 # Use bear methods
22 teddy.eat("fish")
23 teddy.sleep(8)
24
25 pooh.eat("honey")
26 pooh.sleep(5)
27
28 # Get energy levels
29 print("Teddy's energy:", teddy.get_energy())
30 print("Pooh's energy:", pooh.get_energy())
```

# 手続き型プログラミングとの比較

```
1  # Data
2  bears = []
3
4  # Functions
5  def create_bear(name):
6      return {"name": name, "energy": 100}
7
8  def bear_eat(bear, food):
9      print(f"{bear['name']} is eating {food}.
10     Yum!")
11     bear['energy'] += 10
12
13  def bear_sleep(bear, hours):
14      print(f"{bear['name']} is sleeping for
15     {hours} hours. Zzz...")
16     bear['energy'] += hours * 5
17
18  def get_energy(bear):
19      return bear['energy']
20
21  # Create bears
22  teddy = create_bear("Teddy")
23  pooh = create_bear("Pooh")
```

```
24  # Bears eat and sleep
25  bear_eat(teddy, "fish")
26  bear_sleep(teddy, 8)
27
28  bear_eat(pooh, "honey")
29  bear_sleep(pooh, 5)
30
31  # Get energy levels
32  print("Teddy's energy:", get_energy(teddy))
33  print("Pooh's energy:", get_energy(pooh))
```

# 手続き型プログラミングとの比較：クロージャの利用

```
1 def create_bear(name):
2     energy = 100 # Local variable
3
4     def eat(food):
5         nonlocal energy
6         print(f"{name} is eating {food}.
7         Yum!")
8         energy += 10
9
10    def sleep(hours):
11        nonlocal energy
12        print(f"{name} is sleeping for
13        {hours} hours. Zzz...")
14        energy += hours * 5
15
16    def get_energy():
17        return energy
18
19    return eat, sleep, get_energy
```

```
18 # Create closures
19 teddy_eat, teddy_sleep, get_teddy_energy =
20     create_bear("Teddy")
21 pooh_eat, pooh_sleep, get_pooh_energy =
22     create_bear("Pooh")
23
24 # Use closures
25 teddy_eat("fish")
26 teddy_sleep(8)
27
28 pooh_eat("honey")
29 pooh_sleep(5)
30
31 # Get energy levels
32 print("Teddy's energy:", get_teddy_energy())
33 print("Pooh's energy:", get_pooh_energy())
```

# まとめ

- ・ **オブジェクト指向**とは、**プログラミングの書き方、まとめ方**に関する**思考法、概念的枠組み、パラダイム**である。
- ・ オブジェクト指向は、技術的に新しい何かを達成するというよりかは、**プログラム書きやすく、そして理解しやすくするための提案**である。
- ・ 必要なければ、オブジェクト指向によってプログラムを作成しなくても良い
- ・ 新たなデータ構造を持つオブジェクトの作成を行う場合は、**class文**を用いて定義する。定義した変数や値のことは**属性**、関数のことは、**メソッド**という。
- ・ クラスを動的に実行するために作成するオブジェクトを**インスタンス**という。
- ・ インスタンスの初期化を行うために定義するメソッドを**コンストラクタ**という。
- ・ Pythonではコンストラクタに**特殊構文の\_\_init\_\_メソッド**を利用する。
- ・ コンストラクタに引数を渡すには**クラスを呼び出す際に( ) 中で指定して行う**。
- ・ インスタンスから呼び出されたメソッドは**第1引数にインスタンス変数を自動的に取得するので、手動で引数は渡さない**。なお、**インスタンス変数の仮引数はselfとするのが暗黙の了解**となっている。