オブジェクト指向 Object Oriented Programming

スーパーオブジェクトからサブオブジェクト サブオブジェクトを個別に動作 メソッドを通してのみ干渉できる ストラクチャ指向 Structured Programmingclass : ひな形・設計図の様なもの method : 動作・操作（関数） property : 属性・特徴（変数）（attribute とも） instance : class によって作られたもの constructor : class 作成時に実行される method

クラス Class

クラスの書き方

In [1]:
```
class MyClass():
    pass
```

In [2]:
```
# Instance
my_class = MyClass()
```

In [3]:
```
print(type(my_class))
```

```
<class '__main__.MyClass'>
```

In [4]:
```
class MyClass():
    name = "Katsumata" # property
```

In [5]:
```
my_class = MyClass()

print(my_class.name)
```

```
Katsumata
```

In [6]:
```
my_class.age = 22
print(my_class.age)
```

```
22
```

コンストラクタ（initメソッド）・インスタンス変数

In [7]:
```
class MyClass():
    # Constructor
    def __init__(self):
        print("クラスをつくったよ！")
```

In [8]:
```
myclass = MyClass()
```

```
クラスをつくったよ！
```

In [9]:
```
class MyClass():
    def __init__(self):
        self.num  = 1       # Instance Variable
        self.name = "Panda" # Instance Variable
```

In [10]:
```
my_class = MyClass()
```

```
print(my_class.num)
print(my_class.name)
```

```
1
Panda
```

In [11]:
```
# Error Pattern
class MyClass():
    def __init__(self):
        num  = 1       # instance variable
        name = "Panda" # instance variable
```

In [12]:
```
# Error Pattern
my_class = MyClass()

print(my_class.num)
print(my_class.name)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~¥AppData¥Local¥Temp¥ipykernel_11684¥1271134083.py in <cell line: 4>()
      2 my_class = MyClass()
      3
----> 4 print(my_class.num)
      5 print(my_class.name)

AttributeError: 'MyClass' object has no attribute 'num'
```

In [13]:
```
class MyClass():
    def __init__(self, num, name):
        self.num  = num
        self.name = name
```

In [14]:
```
num  = 3
name = "Turtle"

my_class = MyClass(num, name)

print(my_class.num)
print(my_class.name)
```

```
3
Turtle
```

メソッド

In [15]:
```
class MyClass():
    # Constructor
    def __init__(self, num, name):
        self.num  = num
        self.name = name

    # Method
    def my_print(self):
        print(f"num = {self.num}, name = {self.name}")
```

In [16]:
```
num  = 5
name = "Giraffe"

my_class = MyClass(num, name)
```

```
my_class.my_print()
```

```
num = 5, name = Giraffe
```

In [17]:
```
class MyClass():
    # Construct
    def __init__(self, num, name):
        self.num  = num
        self.name = name

    # Method
    def my_print(self):
        print(f"num = {self.num}, name = {self.name}")

    def my_add(self, add_num):
        self.num += add_num
```

In [18]:
```
num  = 5
name = "Giraffe"
add  = 2

my_class = MyClass(num, name)

my_class.my_add(add)
my_class.my_print()
```

```
num = 7, name = Giraffe
```

In [19]:
```
class MyClass():
    def __init__(self, num, name):
        self.num  = num
        self.name = name

    def my_print(self):
        print(f"num = {self.num}, name = {self.name}")

    def my_add(self, add_num):
        self.num += add_num

    def my_zip(self):
        my_list = [self.name, self.num]
        return my_list
```

In [20]:
```
num  = 8
name = "octopus"
add  = 2

my_class = MyClass(num, name)

my_list = my_class.my_zip()
print(my_list)
```

```
['octopus', 8]
```

タイプ

In [21]:
```
class MyClass():
    def __init__(self, num, name):
        self.num  = num
        self.name = name
```

```
    def my_print(self):
        print(f"num = {self.num}, name = {self.name}")

    def my_add(self, add_num):
        self.num += add_num

    def my_zip(self):
        my_list = [self.name, self.num]
        return my_list
```

In [22]:
```
num  = 8
name = "octopus"
add  = 2

my_class = MyClass(num, name)
```

In [23]:
```
print(type(my_class))
print(type(my_class.num))
print(type(my_class.name))
print(type(my_class.my_zip))
```

```
<class '__main__.MyClass'>
<class 'int'>
<class 'str'>
<class 'method'>
```

特殊メソッド

__new__, __init__ __int__, __str__, __len__ __add__, __sub__

In [24]:
```
class MyClass():
    def __init__(self, num):
        self.num = num

    def __float__(self):
        return float(self.num)

    def __add__(self, other):
        return self.num + other.num

    def __eq__(self, other):
        return self.num == other.num
```

In [25]:
```
a = MyClass(6)

print(float(a))
```

```
6.0
```

In [26]:
```
a = MyClass(6)
b = MyClass(3)

print(a + b)
```

```
9
```

In [27]:
```
c = a + b
print(c)
print(type(c))
```

```
9
<class 'int'>
```

In [28]:
```
a = MyClass(6)
b = MyClass(3)

print(a == b)
```

```
False
```

In [29]:
```
class MyClass():
    def __init__(self, num1, num2):
        self.num1 = num1
        self.num2 = num2

    def __add__(self, other):
        num_1 = self.num1 + other.num1
        num_2 = self.num2 + other.num2

        return MyClass(num_1, num_2)
```

In [30]:
```
a = MyClass(5, 6)
b = MyClass(2, 3)

c = a + b

print(c)
print(c.num1, c.num2)
```

```
<__main__.MyClass object at 0x000001F00745DE50>
7 9
```

クラス継承

継承 inheritance

In [31]:
```
class Car():
    def __init__(self, name, speed):
        self.name  = name
        self.speed = speed

    def info(self):
        print(f"name  = {self.name}")
        print(f"speed = {self.speed}")

    def accelerate(self):
        self.speed += 10

    def decelerate(self):
        self.speed -= 10
```

In [32]:
```
prius = Car("prius", 0)
prius.info()

prius.accelerate()
prius.accelerate()
prius.accelerate()
prius.info()
```

```
name  = prius
speed = 0
name  = prius
speed = 30
```

In [33]:
```
# inheritance
class Vehicle(Car):
    pass
```

In [34]:
```
aqua = Vehicle("aqua", 0)
aqua.info()
```

```
name  = aqua
speed = 0
```

スーパークラスを用いた継承・オーバーライド Override

In [35]:
```
class Car():
    def __init__(self, name, speed):
        self.name  = name
        self.speed = speed

    def info(self):
        print(f"name  = {self.name}")
        print(f"speed = {self.speed}")

    def accelerate(self):
        self.speed += 10

    def decelerate(self):
        self.speed -= 10
```

In [36]:
```
# Override
class GasolineVehicle(Car):
    def __init__(self, name, speed, gasoline):
        super().__init__(name, speed)
        # super(GasolineVehicle, self).__init__(name, speed)
        self.gasoline = gasoline

    # Renew
    def info(self):
        print(f"name     = {self.name}")
        print(f"speed    = {self.speed}")
        print(f"gasoline = {self.gasoline}")

    def accelerate(self):
        self.speed    += 10
        self.gasoline -= 50
```

In [37]:
```
yaris = GasolineVehicle("yaris", 0, 1000)
yaris.info()

yaris.accelerate()
yaris.accelerate()
yaris.accelerate()
yaris.info()
```

```
name     = yaris
speed    = 0
gasoline = 1000
name     = yaris
speed    = 30
gasoline = 850
```

In [38]:
```
yaris.decelerate()
yaris.info()
```

```
name     = yaris
speed    = 20
gasoline = 850
```

ポリモーフィズム Polymorphism

In [39]:
```
class ElectricVehicle(Car):
    def __init__(self, name, speed, buttery):
        super().__init__(name, speed)
        self.buttery = buttery

    # Renew, Override
    def info(self):
        print(f"name     = {self.name}")
        print(f"speed    = {self.speed}")
        print(f"buttery  = {self.buttery}")

    def accelerate(self):
        self.speed   += 10
        self.buttery -= 10

    def decelerate(self):
        self.speed   -= 10
        self.buttery += 5
```

In [40]:
```
prius = ElectricVehicle("prius", 0, 100)
prius.info()

prius.accelerate()
prius.accelerate()
prius.accelerate()
prius.info()

prius.decelerate()
prius.info()
```

```
name     = prius
speed    = 0
buttery  = 100
name     = prius
speed    = 30
buttery  = 70
name     = prius
speed    = 20
buttery  = 75
```

In [41]:
```
yaris = GasolineVehicle("yaris", 0, 1000)
prius = ElectricVehicle("prius", 0, 100)

def info(name1, name2):
    name1.info()
    name2.info()
```

```
def accelarate(name):
    name.accelerate()
```

In [42]:
```
info(yaris, prius)
```

```
name     = yaris
speed    = 0
gasoline = 1000
name     = prius
speed    = 0
buttery  = 100
```

In [43]:
```
accelarate(yaris)
info(yaris, prius)
```

```
name     = yaris
speed    = 10
gasoline = 950
name     = prius
speed    = 0
buttery  = 100
```

カプセル化

In [44]:
```
class ElectricVehicle(Car):
    def __init__(self, name, speed, buttery):
        super().__init__(name, speed)
        self.__buttery = buttery

    def info(self):
        print(f"name     = {self.name}")
        print(f"speed    = {self.speed}")
        print(f"buttery  = {self.__buttery}")

    def accelerate(self):
        self.speed   += 10
        self.__buttery -= 10

    def decelerate(self):
        self.speed   -= 10
        self.__buttery += 5
```

In [45]:
```
prius = ElectricVehicle("prius", 0, 100)
prius.info()
```

```
name     = prius
speed    = 0
buttery  = 100
```

In [46]:
```
prius.name = "Prius"
prius.info()
```

```
name     = Prius
speed    = 0
buttery  = 100
```

In [47]:
```
# Error
prius.__buttery -= 10
prius.info()
```

```
---------------------------------------------------------------------------
--
AttributeError                            Traceback (most recent call last)
~¥AppData¥Local¥Temp¥ipykernel_11684¥2052600173.py in <cell line: 2>()
      1 # Error
----> 2 prius.__buttery -= 10
      3 prius.info()

AttributeError: 'ElectricVehicle' object has no attribute '__buttery'
```

In [ ]: