Week 4. Class(객체에 관하여..)

객체(Object)

- 프로그램이 어떤 작업을 수행하기 위해서는 두 가지 요소가 필요
  - (1) 데이터와
  - (2) 데이터를 조작하는 행위
- 일반적으로 데이터는 변수(variable)에 넣어서 사용하고 데이터를 조작하는 일은 함수(function)로 구성해서 쉽게 실행시킬 수 있도록 만들어 놓죠.

객체(object)는 서로 연관된 데이터와 그 데이터를 조작하기 위한 함수를 하나의 집합에 모아놓은 것을 말한다.

객체 지향 프로그래밍을 사용하면 우리가 일상생활에서 생각하는 것과 비슷한 논리로 프로그램을 작성할 수 있기 때문에 프로그램 개발과 유지보수가 간편

Class

뽑기틀, 붕어빵틀 ....

별 모양의 틀(클래스)로 찍으면 별 모양의 뽑기(객체)가 생성되고 하트 모양의 틀(클래스)로 찍으면 하트 모양의 뽑기(객체)가 나오는 것이다.

- 클래스란 똑같은 무엇인가를 계속해서 만들어낼 수 있는 설계 도면 같은 것이고(뽑기 틀), 객체란 클래스에 의해서 만들어진 피조물(별 또는 하트가 찍힌 뽑기)을 뜻한다.

선언

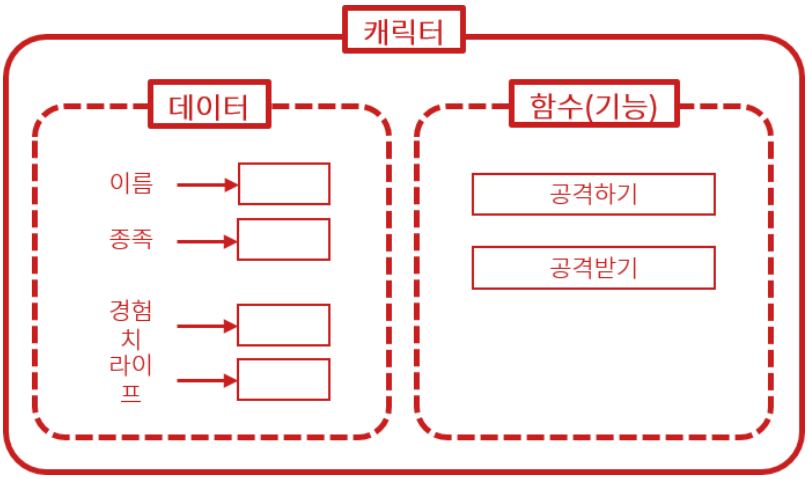
```
class 클래스이름:
    pass
```

캐릭터를 구성하는 데이터를 생각해보면

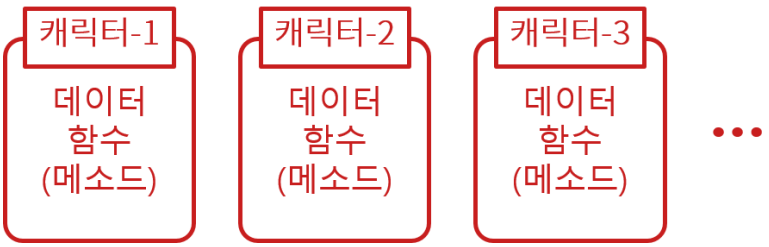
- 이름
- 종족
- 라이프
- 경험치

캐릭터과 관련된 기능을 생각해보면

- 공격하기
- 공격받기



객체 지향 프로그래밍이란 명함을 구성하는 데이터와 명함과 관련된 함수를 묶어서 명함이라는 새로운 객체(타입)를 만들고 이를 사용해 프로그래밍하는 방식을 의미



클래스 정의

In [3]:

```
class Character:
    name = 'Default'
```

객체 생성

In [4]:

```
a = Character()
print(type(a))

a.name = "마포구보안관"
print('a 객체의 이름은:', a.name)

b = Character()
b.name = "한조"
print('b 객체의 이름은:', b.name)
```

```
<class '__main__.Character'>
a 객체의 이름은: 마포구보안관
b 객체의 이름은: 한조
```

self 살펴보기

캐릭터가 공격을 받을 때, 생성한 캐릭터가 함수의 input으로 들어가야했다

`attacked` 라는 함수의 첫 번째 인수값으로 self라는 것을 사용했다.

> 메서드의 첫 번째 인자는 항상 self여야 한다
>
> 또한 메서드(함수)의 입력인자(input)가 하나도 없으면 안됨
>
> 하나도 없다면 self라도 전달해줘야해

In [1]:

```
class Character:
    name = None        # 처음 object를 생성하면 이름이 없죠
    life = 100         # 초기 라이프는 100

    def attacked(self):
        self.life -= 10 # 공격을 받으면 life에서 10씩 감소함
        print('%s 이(가) 공격을 받아 생명력이 %d 로 줄었습니다.' %(self.name, self.life))
```

In [2]:

```
b = Character()
b.name = "한조"
print(b.life)
b.attacked()

a = Character()
a.name ="메르시"
a.attacked()
a.attacked()
a.attacked()
```

```
100
한조 이(가) 공격을 받아 생명력이 90 로 줄었습니다.
메르시 이(가) 공격을 받아 생명력이 90 로 줄었습니다.
메르시 이(가) 공격을 받아 생명력이 80 로 줄었습니다.
메르시 이(가) 공격을 받아 생명력이 70 로 줄었습니다.
```

실제로는 위 함수를 사용하기 위해서는 `b.attacked(b)` 로 사용해야한다.

- 그런데 이전 함수를 호출할 때 아래 문장처럼 b를 따로 전달하여 사용하지 않아도 문제가 없었다. 왜 그럴까?

`b.attacked()` 이라는 호출이 발생하면 attacked 함수의 첫번째 인수인 self에는 호출 시 이용했던 객체(즉, b라는 아이디)가 자동으로 전달된다.

객체변수

In [5]:

```
class Character:
    life = 100

    def setname(self, inputname): #setname 의 input 인자는 name
        self.name = inputname #본 객체의 변수인 name에 input으로 받았던 이름을 넣어줌
        print('이 캐릭터의 이름이 %s 로 정해졌습니다.' %(self.name))

    def attacked(self):
        self.life -= 10 # 공격을 받으면 life에서 10씩 감소함
        print('%s 이 공격을 받아 생명력이 %d 로 줄었습니다.' %(self.name, self.life))
```

In [6]:

```
c = Character()
c.setname('루시우')
c.attacked()
```

```
이 캐릭터의 이름이 루시우 로 정해졌습니다.
루시우 이 공격을 받아 생명력이 90 로 줄었습니다.
```

"c 라는 아이디를 가진 사람이 자신의 이름을 `루시우` 으로 설정하려고 하는구나. 그렇다면 앞으로 c라는 아이디로 접근하면 이 사람의 이름이 `루시우` 이라는 것을 잊지 말아야겠다."

위와 같이 c 라는 아이디와 `루시우` 이라는 이름을 연결해 주는 것이 바로 self이다.

setname함수가 실행되는 순서는 다음과 같다.

1. c라는 아이디를 가진 사람이 "루시우"이라는 이름을 setname 함수에 입력으로 준다.

   c.setname("루시우")

2. 그러면 다음의 문장이 수행된다.

   self.name = inputname

3. self는 setname 함수의 첫 번째 입력값으로 c라는 아이디가 자동으로 전달되므로 다음과 같이 바뀔 것이다.

> c.name = name

4. name은 setname 함수의 두 번째로 입력받은 "한국인"이라는 값이므로 위의 문장은 다시 다음과 같이 바뀔 것이다.

> c.name = "루시우"

객체변수 수정하기

객체 변수는 다음과 같이 생성할 수 있다.

`객체.객체변수 = 값`

In [7]:

```
c.name = "홍길동"
```

> 일반적으로 객체의 속성(attribute)를 수정을 필요로 할 때는 함수를 만들어서 제어한다

__init__ 이란 무엇인가?

위 예제에서 아래 코드를 실행 해 보면

```
AttributeError: 'Character' object has no attribute 'name'
```

이라는 오류가 발생함.

> 이는 attacked() 함수를 실행하는데에 self.name 객체변수를 필요로 하지만, setname()함수를 먼저 실행 해 주지 않으면 name 이라는 변수가 존재하지 않음.

이게 무슨 말이냐면

> - 붕어빵으로 비유해 보면 붕어빵 틀(클래스)을 이용해
> - 팥소를 넣지 않은 상태로 붕어빵을 구운 후(인스턴스 생성)
> - 나중에 다시 붕어빵 안으로 팥소를 넣는 것

In [8]:

```
babo = Character()
babo.attacked()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-8-f08b3fc8fe1a> in <module>()
      1 babo = Character()
----> 2 babo.attacked()

<ipython-input-5-691b47894de7> in attacked(self)
      8     def attacked(self):
      9         self.life -= 10 # 공격을 받으면 life에서 10씩 감소함
---> 10         print('%s 이 공격을 받아 생명력이 %d 로 줄었습니다.' %(self.nam
e, self.life))

AttributeError: 'Character' object has no attribute 'name'
```

이를 방지하기 위해서, `__init__` 이라는 함수를 생성함.

이 `__init__`이라는 함수는 특별한 의미를 갖는데, 그 의미는

"객체를 만들 때 항상 실행된다."

즉, 아이디를 부여받을 때 항상 실행된다는 말이다. 따라서 이제는 캐릭터를 만들려면 다음처럼 입력해야 한다.

In [9]:

```
class Character:
    life = 100 # 모든 Character 객체는 life 가 100 을 가짐

    def __init__(self, name): #setname 의 input 인자는 name
        self.name = name #본 객체의 변수인 name에 input으로 받았던 이름을 넣어줌
        print('이 캐릭터의 이름이 %s 로 정해졌습니다.' %(self.name))

    def attacked(self):
        self.life -= 10 # 공격을 받으면 life에서 10씩 감소함
        print('%s 이 공격을 받아 생명력이 %d 로 줄었습니다.' %(self.name, self.life))

    def get_info(self):
        print('이름: %s, 라이프: %d' %(self.name, self.life))
```

In [10]:

```
d = Character('한조')
d.attacked()
d.attacked()
d.attacked()
d.get_info()
```

```
이 캐릭터의 이름이 한조 로 정해졌습니다.
한조 이 공격을 받아 생명력이 90 로 줄었습니다.
한조 이 공격을 받아 생명력이 80 로 줄었습니다.
한조 이 공격을 받아 생명력이 70 로 줄었습니다.
이름: 한조, 라이프: 70
```

> 이제는 `__init__` 함수 때문에 d = Character("한조")처럼 객체(캐릭터)를 생성 할 때 이름까지 함께 입력해야 한다.

특수 Method

- 위에서 살펴 본 것과 같이 `__init__` 의 경우는 모양이 특이함.
- 앞 뒤로 `__` 가 붙어 있는데 이는 python method 들의 특수한 메서드임.

변수 이름을 치고 엔터 키(주피터 노트북의 경우에는 시프트 + 엔트)를 치면 변수의 값이 호출되는데 사실 이것은 해당 변수가 가지는 `__repr__` 이라는 메서드가 호출되는 것이다.

repr은 representation의 약자이다. 또 변수를 str이라는 함수에 넣으면 변수를 문자열로 변환해 주는데 이것도 사실은 `__str__` 이라는 메서드가 호출되는 것이다.

In [11]:

```
# 위에서 만들었던 캐릭터 'd'
d
```

Out[11]:

```
<__main__.Character at 0x4e02cd0>
```

```
def __repr__(self):
    return "캐릭터: 이름 = %s 체력 = %d" % (self.name, self.life)
```

In [12]:

```
class Character:
    life = 100 # 모든 Character 객체는 life 가 100 을 가짐

    def __init__(self, name): #setname 의 input 인자는 name
        self.name = name #본 객체의 변수인 name에 input으로 받았던 이름을 넣어줌
        print('이 캐릭터의 이름이 %s 로 정해졌습니다.' %(self.name))

    def attacked(self):
        self.life -= 10 # 공격을 받으면 life에서 10씩 감소함
        print('%s 이 공격을 받아 생명력이 %d 로 줄었습니다.' %(self.name, self.life))

    def __repr__(self):
        return "캐릭터 - 이름 = %s 체력 = %d" % (self.name, self.life)
```

In [13]:

```
# 새로 representation을 부여받은 아이 'e'
e = Character('Beta')
print(e)
```

```
이 캐릭터의 이름이 Beta 로 정해졌습니다.
캐릭터 - 이름 = Beta 체력 = 100
```

Class 자세히 알기

객체를 만들어내는 공장과도 같다. 이 객체를 어떻게 사용할 수 있는지를 알려면 클래스의 구조를 보면 된다. 즉, 클래스는 해당 객체의 청사진(설계도)이라고 할 수 있다.

> 클래스는 무작정 만들기 보다는 클래스에 의해서 만들어진 객체를 중심으로 어떤 식으로 동작하게 할 것인지 미리 구상을 한 후에 생각했던 것들을 하나씩 해결하면서 완성해 나가는 것이 좋다.

클래스변수와 객체(인스턴스) 변수

초보자들이 많이 어려워하는 개념 중 하나인 클래스 변수(class variable)와 인스턴스 변수(instance variable)에 대해 살펴보겠다

```
class Character:
    num_characters = 0
    life = 100 # 모든 Character 객체는 life 가 100 을 가짐

    def __init__(self, name): #setname 의 input 인자는 name
        Character.num_characters += 1
        self.name = name #본 객체의 변수인 name에 input으로 받았던 이름을 넣어줌
        print('이 캐릭터의 이름이 %s 로 정해졌습니다.' %(self.name))

    def attacked(self):
        self.life -= 10 # 공격을 받으면 life에서 10씩 감소함
        print('%s 이 공격을 받아 생명력이 %d 로 줄었습니다.' %(self.name,
self.life))
```

여기서 보면 `num_characters` 는 `Character` 라는 클래스에 속하는 클래스 변수이고,

`self.name` 과 같이 `name` 의 경우는 각 클래스에서 생성되는 인스턴스마다 다른 값을 가지므로 인스턴스 변수이다.

그럼 `life` 는 무엇일까?...

사실 `__init__` 부분에 들어가는 것이 더 옳은 변수이지만 클래스 매소드에서 `self.life` 처럼 인스턴스마다 달라지는 변수이므로 이는 인스턴스 변수라고 하는 것이 좋음

그렇다면 언제 클래스 변수를 사용해야 하고 언제 인스턴스 변수를 사용해야 할까?

여러 인스턴스 간에 서로 공유해야 하는 값은 클래스 변수를 사용해야 함. 왜냐하면 파이썬은 객체(인스턴스)의 네임스페이스(정보)에 없는 이름은 클래스 자체의 네임스페이스(정보)에서 찾아보기 때문에,

이러한 특성을 이용하면 `클래스 변수가 모든 객체에 공유` 될 수 있습니다

In [15]:

```
class Character:
    num_characters = 0 # 모든 캐릭터의 개체 수

    def __init__(self, name): #setname 의 input 인자는 name
        self.life = 100 # 모든 Character 객체는 life 가 100 을 가짐
        self.name = name #본 객체의 변수인 name에 input으로 받았던 이름을 넣어줌
        print('이 캐릭터의 이름이 %s 로 정해졌습니다.' %(self.name))
        Character.num_characters += 1

    def attacked(self):
        self.life -= 10 # 공격을 받으면 life에서 10씩 감소함
        print('%s 이 공격을 받아 생명력이 %d 로 줄었습니다.' %(self.name, self.life))

    @classmethod
    def how_many_character(cls):
        print ("생성된 총 캐릭터 수:", cls.num_characters)
```

In [16]:

```
a = Character('한조')
b = Character('겐지')
c = Character('맥크리')
d = Character('리퍼')
e = Character('루시우')

print("생성된 총 캐릭터의 수 :",Character.num_characters)
Character.how_many_character()
```

```
이 캐릭터의 이름이 한조 로 정해졌습니다.
이 캐릭터의 이름이 겐지 로 정해졌습니다.
이 캐릭터의 이름이 맥크리 로 정해졌습니다.
이 캐릭터의 이름이 리퍼 로 정해졌습니다.
이 캐릭터의 이름이 루시우 로 정해졌습니다.
생성된 총 캐릭터의 수 : 5
생성된 총 캐릭터 수: 5
```

메서드 `how_many_character` 는 객체에 소속되어 있지 않고 클래스에 소속되어 있는 메서드임.

그렇다면 언제 클래스 메서드를 사용해야 할까?

> 여기 서 우리가 해당 클래스의 어떤 부분까지 알아야 할 지에 따라 메소드를 클래스 메소드(class mathod) 로 정의할지 스태틱 메소드(static method) 로 정의할지 결정할 수 있습니다.

여기서는 how_many 메소드를 클래스 메소드로 만들어 주기 위해 `데코레이터` 를 이용하였습니다.

데코레이터는 어떤 일을 추가로 해 주는 더 큰 함수로 해당 부분을 감싸주는 것이라고 생각하면 됩 니다. 즉, `@classmethod` 데코레이터가 사용된 것.

클래스의 상속

```
class 자식클래스이름(부모클래스이름):

    def __init__(self, 속성값1, 속성값2):
        super().__init__(속성값, 속성값2)
        자식 클래스의 초기화 코드
```

- 클래스를 만들다 보면 상속받을 대상인 클래스의 메서드와 이름은 같지만 그 행동을 다르게 해야 할 때가 있다.
-
- 상속(Inheritance)이란 "물려받다"라는 뜻으로, "재산을 상속받다"라고 할 때의 상속과 같은 의미이다. 클래스에도 이런 개념을 적용할 수가 있다.
- 어떤 클래스를 만들 때 다른 클래스의 기능을 물려받을 수 있게 만드는 것이다.
- 클래스 상속을 사용하면 이미 만들어진 클래스 코드를 재사용하여 다른 클래스를 생성할 수 있다
- 이 때 상속을 받는 클래스를 자식 클래스(child class), 상속의 대상이 되는 클래스를 부모 클래스(parent class)라고 한다.

만약 각 캐릭터들은 서로 다른 초기 속성값을 가지고 태어난다면

In [17]:

```
class BossCharacter(Character): # 여기서는 위에 만들었던 Character 클래스를 상속 받음)
    def __init__(self,name):
        super().__init__(name)
        self.life = 1000 # Character 클래스와 같고 life 만 1000인 보스몹을 만들었다.
```

In [18]:

```
boss = BossCharacter('보스다')
print('보스의 라이프는', boss.life)
boss.attacked()
boss.attacked()
boss.attacked()
```

```
이 캐릭터의 이름이 보스다 로 정해졌습니다.
보스의 라이프는 1000
보스다 이 공격을 받아 생명력이 990 로 줄었습니다.
보스다 이 공격을 받아 생명력이 980 로 줄었습니다.
보스다 이 공격을 받아 생명력이 970 로 줄었습니다.
```

메서드의 오버라이딩

클래스를 만들다 보면 상속받을 대상인 클래스의 메서드와 이름은 같지만 그 행동을 다르게 해야 할 때가 있다.

여기서 보스몹(BossCharacter) 클래스가 상속받은 attacked 함수를 Character 클래스의 그것과 다르게 동작하도록 하려한다.

In [19]:

```
class BossCharacter(Character): # 여기서는 위에 만들었던 Character 클래스를 상속 받음)
    def __init__(self,name):
        super().__init__(name)
        self.life = 1000 # Character 클래스와 같고 life 만 1000인 보스몹을 만들었다.

    def attacked(self):
        self.life -= 1 # 공격을 받으면 life에서 10씩 감소함
        print('%s 이(가) 공격을 받았지만 생명력이 %d 로 조금 줄었습니다.' %(self.name, self.lif
```

In [20]:

```
boss = BossCharacter('보스2')
boss.attacked()
```

이 캐릭터의 이름이 보스2 로 정해졌습니다.
보스2 이(가) 공격을 받았지만 생명력이 999 로 조금 줄었습니다.

In [21]:

```
class Parent:
    house = "yong-san"
    def __init__(self):
        self.money = 10000

class Child1(Parent):
    def __init__(self):
        super().__init__()
        pass

class Child2(Parent):
    def __init__(self):
        pass

child1 = Child1()
child2 = Child2()
print(child1.money)
print(child1.house)
print(child2.house)
print(child2.money)
```

```
10000
yong-san
yong-san

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-21-db34367a9e0f> in <module>()
     18 print(child1.house)
     19 print(child2.house)
---> 20 print(child2.money)

AttributeError: 'Child2' object has no attribute 'money'
```

SIMPLE!

여기 캐릭터들은 공격만 받을 줄 알지 ... 공격 좀 해보고 싶다

Character 클래스를 수정을 좀 하면

In [41]:

```
class Character:
    num_characters = 0 # 모든 캐릭터의 개체 수

    def __init__(self, name): # __init__의 input 인자는 name
        self.name = name #본 객체의 변수인 name에 input으로 받았던 이름을 넣어줌
        self.life = 100 # 모든 Character 객체는 life 가 100 을 가짐
        self.exp = 0 # 경험치는 0부터 시작
        self.__hidden_dps = 0.5 # 히든 dps
        self.dps = 0.5 # 눈에 보에는 dps
        print('이 캐릭터의 이름이 %s 로 정해졌습니다.' %(self.name))
        Character.num_characters += 1

    def attacked(self):
        self.life -= 10 # 공격을 받으면 life에서 10씩 감소함
        print('%s 이 공격을 받아 생명력이 %d 로 줄었습니다.' %(self.name, self.life))

    def attack(self, other):
        self.exp += 50 # 경험치 50 증가
        print('%s 가 %s 를 공격해서 경험치가 50 올랐습니다.' %(self.name, other.name))

    def __add__(self, other):
        print("%s 와 %s 가 파티를 맺었습니다." % (self.name, other.name))
```

In [42]:

```
user1 = Character('유저1')
user2 = Character('유저2')

print("총 캐릭터의 수:", Character.num_characters)
```

```
이 캐릭터의 이름이 유저1 로 정해졌습니다.
이 캐릭터의 이름이 유저2 로 정해졌습니다.
총 캐릭터의 수: 2
```

In [43]:

```
user1.attack(user2)
user2.attacked()
print('유저1의 경험치:', user1.exp)
```

```
유저1 가 유저2 를 공격해서 경험치가 50 올랐습니다.
유저2 이 공격을 받아 생명력이 90 로 줄었습니다.
유저1의 경험치: 50
```

In [44]:

```
user2.attack(user1)
user2.attack(user1)
print('유저2의 경험치:', user2.exp)

print('--------------------')
user2 + user1
```

```
유저2 가 유저1 를 공격해서 경험치가 50 올랐습니다.
유저2 가 유저1 를 공격해서 경험치가 50 올랐습니다.
유저2의 경험치: 100
--------------------
유저2 와 유저1 가 파티를 맺었습니다.
```

> 모든 클래스 속성은 클래스 외부에 공개되어 있습니다.

한가지 예외가 있는데,

밑줄 두 개 로 시작하는 데이터 속성을 정의하면, 즉 예를 들어 `__name` 과 같이 하면, 파이썬이 이것을 클래스 외부로 드러나지 않도록 숨겨 줍니다.

In [46]:

```
print(user1.dps)
print(user1.__hidden_dps)
```

```
0.5

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-46-33d0574581e3> in <module>()
      1 print(user1.dps)
----> 2 print(user1.__hidden_dps)

AttributeError: 'Character' object has no attribute '__hidden_dps'
```