

[Python] 파이썬 기초 11 - 객체의 4대 특성 (상속화, 캡슐화, 다형성, 추상화) — 나무
늘보의 개발 블로그

노트북: 첫 번째 노트북

만든 날짜: 2020-10-24 오후 7:19

URL: <https://continuous-development.tistory.com/71?category=736681>

Python

[Python] 파이썬 기초 11 - 객체의 4대 특성 (상속화, 캡슐화, 다형성, 추상화)

2020. 8. 18. 23:47 수정 삭제 공개

객체의 4대 특성

1.<캡슐화>

말 그대로 캡슐로 보호하듯이 외부에서 데이터에 대한 접근을 막는 것이다.

이것을 인캡슐레이션이라고 한다. 데이터에 대한 접근을 get / set으로 만 하게끔 한다.

변수명 앞에 _ 이것을 두 개 붙이면 __ private 개념으로 적용된다. 해당 class? 공간에서만 변수를 건들 수 있고 이외에서는 건들 수 없다.

__ 이게 없을 때는 public 개념이다. 모든 곳에서 해당 변수를 사용할 수 있다.

```
functionMain.py × first_class_fuction.py × hmsMain.py × employee.py × inheritance.py ×
1
2
3 # encapsulation (은닉화)
4 # information - hiding (정보은닉)
5
6 # 변수명에 __ 이것을 통해 접근자체를 못하게 막는다. 이 __가 없다는건 public이라는 뜻이다. 함수에도 가
7 # 함수를 통해서만 값을 얻어갈 수 있다. 인스턴스 소유 이지만 마음대로 볼 수 없다.
8 class MyDate(object):
9
10     def setYear(self, year):
11         if(year < 0):
12             self.__year = 2020
13         else:
14             self.__year = year
15     def getYear(self):
16         return self.__year
17
```

```
package_function.py × oop_first_class.py × student.py × emp_caller.py × inheritance_caller.py ×
1
2 from service.oop.hms.inheritance import *
3
4 mydate = MyDate()
5 mydate.setYear(2010) # set으로 설정을 하는데 if문 조건에 년도에 - 값이 나옴 2020을 출력하게끔 한다.
6 print(mydate.getYear()) # 설정해놓은 값을 가져온다.
```

이렇게 get / set을 제외하고는 해당 변수를 다시 설정할 수 없는 식으로 만든다. 저 year값을 caller 부분에서 임의로 꺼낼 수 없다.
get에서는 값을 가져오고 set에서는 값을 재설정한다. 이렇게 값을 설정한다.

2.<상속> + <다형성>

부모 클래스에서는 자식클래스에 있는 변수나 함수를 사용하지 못하지만 자식클래스는 부모클래스에 대한 접근이 가능하다. 그래서 부모에 따른

객체를 생성하지 않더라도 자식 클래스에서는 부모 클래스를 아무런 제약 없이 접근해서 쓸 수 있다.

오버 라이딩 - 자식이 부모 함수를 다시 재정의해서 사용한다. 상속에서만 나오는 개념이다.

오버 로딩 - 부모의 함수의 메서드에 매개변수나 / 타입 / 리턴 값을 다르게 줘 사용한다.

부모가 가진걸 자식 쪽에서 상속받아서 사용한다.

```
21 # inheritance
22
23 class Sup(object):
24     def __init__(self, name):
25         self.name = name
26
27     def getState(self):
28         print('Super - ', self.name)
29
30
31 class Sub(Sup): # 클래스를 받는 것 자체로 상속이 된다.
32     def getState(self):
33         print('Sub - ', self.name) # 오버라이딩 - 부모의 정의된 메소드를 재정의 한다. 이것이 다형성
34
35
```

상속은 클래스의 매개변수를 클래스로 받는 것으로 상속을 받는다.

```
package_function.py × oop_first_class.py × student.py × emp_caller.py × inheritance_caller.py ×
1
2 from service.oop.hms.inheritance import *
3
4 mydate = MyDate()
5 mydate.setYear(2010) # set으로 설정을 하는데 if문 조건에 년도에 - 값이 나옴 2020을 출력하게끔 한다.
6 print(mydate.getYear()) # 설정해놓은 값을 가져온다.
7
8 #Sub에는 default가 없고 매개 변수 하나짜리 받는게 없어도 되는 이유가 상속을 받고 있어서다. 그래서
9 # 부모의 생성자가 호출 됐다.
10 sub = Sub('상속관계 아들 생성자 호출') # sub
11
12 sub.getState() # 부모 함수를 사용 가능하다.
```

객체를 생성한 후 부모가 가지고 있는 함수를 사용할 수 있다. (getState
())

```

20
21 # inheritance
22
23 class Sup(object):
24     def __init__(self, name):
25         self.name = name
26
27     def getState(self):
28         print('Super - ', self.name)
29
30
31 class Sub(Sup): # 클래스를 받는 것 자체로 상속이 된다.
32     # def getState(self):
33     #     print('Sub - ', self.name) # 오버라이딩 - 부모의 정의된 메소드를 재정의 한다. 이것이 다형성
34     pass
35

```

```

package_function.py × oop_first_class.py × student.py × emp_caller.py × inheritance_caller.py ×
1
2 from service.oop.hms.inheritance import *
3
4 mydate = MyDate()
5 mydate.setYear(2010) # set으로 설정을 하는데 if문 조건에 년도에 - 값이 나옴 2010을 출력하게끔 한다.
6 print(mydate.getYear()) # 설정해놓은 값을 가져온다.
7
8 #Sub에는 default 가 없고 매개 변수 하나짜리 받는게 없어서 되는 이유가 상속을 받고 있어서다. 그래서
9 # 부모의 생성자가 호출 됐다.
10 sub = Sub('상속관계 아들 생성자 호출') # sub
11
12 sub.getState() # 부모 함수를 사용 가능하다.

```

부모 생성자 호출하는 경우

```
functionMain.py × first_class_fuction.py × hmsMain.py × employee.py × inheritance.py ×
67         return super().perInfo() + ',' + self.subject
68
69
70 class Car(object):
71     def __init__(self, speed):
72         self.speed = speed
73
74     def getSpeed(self):
75         return self.speed
76
77     def carDesc(self):
78         return "차량 : {}".format(self.speed)
79
80
81 class SportsCar(Car):
82     def __init__(self, speed, turbo): #선언은 해주되
83         super().__init__(speed) # 여기서 부모의 껍로 상속을 받아 가져온다.
84         self.turbo = turbo
85
86     def getTurbo(self):
87         return self.turbo
88
89     def carDesc(self):
90         return super().carDesc() + " | 터보 여부 = {}".format(self.turbo) # 오버라이딩(재정의)
91
92
93 class Truck(Car):
94     def __init__(self, speed, capacity):
95         super().__init__(speed)
96         self.capacity = capacity
```

이렇게 부모의 생성자를 호출해서 사용 가능하다.

사용할 때는 super()를 통해 가져오고 super().__init__ 부모의 생성자인 __init__을 가져온다. 그 후 해당 매개변수를 넣어준다.

그리고 자신의 변수는 따로 self로 정의해준다.

```

package_function.py x oop_first_class.py x student.py x emp_caller.py x inheritance_caller.py x
15 tea01 = TeacherV0(' 쌤쌤', 48, '부산', '파이썬')
16
17
18 print(stu01.perInfo())
19 print(tea01.perInfo())
20
21 # perList = [stu01, tea01]
22 perList = []
23 perList.append(stu01)
24 perList.append(tea01)
25 for obj in perList: # 저 조건에 타입이 많아진다면 if else 수백개가 된다.
26     if isinstance(obj, StudentV0): # isinstance - 주어진 인스턴스가 특정 클래스/데이터 타입인지 검사해주는 함수
27         print(obj.stuInfo())
28     else:
29         print(obj.teaInfo())
30
31
32 scCar = SportsCar(300, '터보기능')
33 print("speed - ", scCar.getSpeed()) # speed - 300
34 print("desc - ", scCar.carDesc()) # desc - 차량 : 300 | 터보 여부 = 터보기능
35 print("desc - ", Car.carDesc(scCar)) # desc - 차량 : 300
36
37 truck = Truck(300, 1000)
38 print("speed - ", truck.getSpeed()) # speed - 300
39 print("desc - ", truck.carDesc()) # desc - 차량 : 300 적재공간 = 1000
40

```

예제문제

```

functionMain.py x first_class_function.py x hmsMain.py x employee.py x inheritance.py x package_function.py x oop_first_class.py x student.py x emp_caller.py x inheritance_caller.py x
196 # [실습]
197 # 1.Account class 작성 - account, balance, interestRate
198 # 2.accountInfo() - 계좌의 정보를 출력한다. (account, balance, interestRate)
199 # 3.deposit(amount) - 계좌 잔액에 매개변수로 들어온 amount 를 누적한다.
200 # 4.printInterestRate() - 현재 잔액에 여자를 계산하여 소수점 자리 2자리까지 출력한다.
201 # 5.withDraw(amount) - 매개변수로 들어온 금액만큼을 출금하여 잔액을 변경한다.
202 # 단) 잔고 부족할 경우 '잔액이 부족하여 출금할 수 없습니다.' 출력한다.
203
204 class Account(object):
205     def __init__(self, account, balance, interestRate):
206         self.account = account
207         self.balance = balance
208         self.interestRate = interestRate
209
210     def accountInfo(self):
211         print('account : {} | balance : {} | interestRate : {}'.format(self.account, self.balance, self.interestRate))
212
213     def deposit(self, amount):
214         self.balance += amount
215
216     def withDraw(self, amount):
217         if (self.balance > int(amount)):
218             self.balance -= amount
219         else:
220             print('잔액이 부족하여 출금할 수 없습니다.')
221
222     def printInterestRate(self):
223         print('이자는 {} 입니다.'.format(round((self.balance*self.interestRate),2)))
224
225
226 truck = Truck(300, 1000)
227 print("speed - ", truck.getSpeed()) # speed - 300
228 print("desc - ", truck.carDesc()) # desc - 차량 : 300 적재공간 = 1000
229
230 print("=====")
231 print("=====")
232 print("=====")
233 print("===== 은행계좌 =====")
234 # caller 쪽에서 객체 생성후
235 account = Account('441-2919-1234', 500000, 0.073)
236 account.accountInfo()
237 account.withDraw(600000)
238 account.deposit(200000)
239 account.accountInfo()
240 account.withDraw(600000)
241 account.accountInfo()
242 print('현재 잔액의 이자를 포함한 금액')
243 account.printInterestRate()
244

```

<다중 상속>

파이썬은 다중상속을 권장하지 않는다.

```
inheritance_caller.py ~ hmsMain.py ~ employee.py ~ inheritance.py ~ exception.py ~ exec.py ~ test123.py ~ inheritance_caller.py ~ exception_caller.py
1 # 다중 상속, 추상화
2
3 class Animal(object):
4     def cry(self):
5         print('크아아아아아')
6
7
8 class Tiger(Animal):
9     def jump(self):
10        print('호랑이가 점프를 한다.')
11
12
13 class Lion(Animal):
14     def bite(self):
15        print('한 입에 꿀꺽한다.')
16     def cry(self):
17        print('그르렁')
18
19
20 class Liger(Tiger, Lion): # 다중상속 - 파냐의 클래스가 2개이상의 클래스를 상속받는다.
21     def play(self):
22        print('라이거가 사육사랑 수영했습니다.')
23
24
25 from service.oop.has.exception import *
26
27 animal = Liger()
28 animal.cry() # 부모클래스 받을 때 tiger를 먼저 받을, 함수가 중복될 때는 먼저 상속 받은 함수를 사용한다.
29 animal.jump()
30 animal.bite()
31 animal.play()
```

```
C:\Users\i\Anaconda3\python.exe C:/Users/i/PycharmProjects/python_base/exception_caller.py
그르렁
호랑이가 점프를 한다.
한 입에 꿀꺽한다.
라이거가 사육사랑 수영했습니다.
Process finished with exit code 0
```

4. 추상화

oop의 마지막 컨셉은 추상화가 있다. 부모에서 자식으로 내려올수록 구체화되고 자식에서 부모쪽으로 올라갈 수록 추상화 된다.

상위클래스의 역할은 자식들에게 퍼주는 역할이다. 부모로서의 역할을 한다.

부모 클래스에서 추상메소드라는 것을 정의 할 수있다.

함수라는것은 반드시 구현부가 있어야한다. 근데 클래스에서는 구현부가 없는 함수를 가질수 있다.

구현부가 없는 클래스는 추상클래스가 되고 이 추상클래스를 상속받는 하위 클래스는 부모의 정의되어 있는 **추상클래스들을 구현화**해야한다 (오버라이딩 해야한다.)

이건 강제성을 띄우고 안할경우 에러를 반환한다.

추상화를 하는이유는 일정한 틀 or 형식을 유지하게끔 한다.

부모 클래스에서 함수 부분에 비어있는 바디를 만들어놓고 자식클래스에게 함수 구현이라는 강제성을 띄운다. 자식들은 이걸 반드시 오버라이딩 해야 된다. 단 함수의 이름은 부모의 이름을 따르게끔 강제한다.

이렇게 강제함으로써 일정한 틀을 유지하게끔 한다. 어느정도의 표준을 제공한다고 보면 될 것 같다.

@abstractmethod 를 사용함으로써 추상클래스를 사용한다는 선언을 한다

이와 더불어 class에서 받는 매개변수를 metaclass = ABCMeta 를 받아 준다. 이런 문법형식을 따른다.

```
# 추상클래스(객체 생성이 불가하다.)
# 메서드의 목록만 가진 클래스
# 상속받는 클래스에서 메서드 구현을 강제하기 위해서 사용하는 문법

from abc import *

class Base(metaclass=ABCMeta): # 추상 클래스에서는 이렇게 매개변수를 받게 해야한다. 문법이다.
    @abstractmethod # 구현부가 없는 추상 메소드라는 선언을 한다.
    def study(self):
        pass

    @abstractmethod # 구현부가 없는 추상 메소드라는 선언을 한다.
    def goToAcademy(self):
        pass

class BaseSub(Base):
    def study(self):
        print('공부하자')

    def goToAcademy(self):
        print('학원을 가자')
```

추상클래스(객체 생성이 불가하다.)

메서드의 목록만 가진 클래스

상속받는 클래스에서 메서드 구현을 강제하기 위해서 사용하는 문법


```
eritance_caller.py × exception_caller.py ×
from service.oop.hms.exception import *

animal = Liger()
animal.cry() # 부모키를 받을 때 tiger를 먼저 받음, 함수가 중복될때는 먼저 상속 받은 함수를 사용한다.
animal.jump()
animal.bite()
animal.play()

# 일반 클래스는 객체 생성이 바로 가능하지만 추상메소드는 인스턴스 생성이 안된다.
# Can't instantiate abstract class Base with abstract methods goToAcademy, study
# 클래스에 대해 객체 생성을 했을때
# base = Base()

# 자식클래스에서 하나만 오버라이딩 했을때
# base = BaseSub() # Can't instantiate abstract class BaseSub with abstract methods goToAcademy
# print(base)

base = BaseSub() # 생성자 호출로 인한 객체 생성
base.study() # 공부하자
```

일반 클래스는 객체 생성이 바로 가능하지만 추상 메소드는 인스턴스 생성이 안된다.

Can't instantiate abstract class Base with abstract methods goToAcademy, study

클래스에 대해 객체 생성을 했을때

base = Base()

자식클래스에서 하나만 오버라이딩 했을때

base = BaseSub() # Can't instantiate abstract class BaseSub with abstract methods goToAcademy

print(base)

추상 메소드에 대한 모든 구현을 해야 객체를 생성 할 수 있다.

'Python' 카테고리의 다른 글

[Python] 파이썬 기초 13 - 파이썬을 통한 파일 입출력 사용법

[Python] 파이썬 기초 12 - 예외처리

[Python] 파이썬 기초 11 - 객체의 4대 특성 (상속화, 캡슐화, 다형성, 추상화)□

[Python] 파이썬 기초 10 - 클래스에 대한 정의와 사용법□

[Python] 파이썬 기초 9 - 패키지과 모듈에 대한 정의와 다양한 함수 형태□

[Python] 파이썬 기초 8 - 반복문(for , while)에 대한 정의와 기본적인 함수 사...

객체 4대 특성

객체의 4대 특성

객체지향의 4대 특징

객체지향의 4대특성

파이썬 4대 특성



나무늘보스

혼자 끄적끄적하는 블로그 입니다.