IPA 주관 인공지능센터 기본(fundamental) 과정

- GitHub link: here
- E-Mail: windkyle7@gmail.com

객체지향 프로그래밍

OOP (Object-Oriented Programming)

객체지향 개념

- **객체 (Object)**
  - 현실 세계에 존재하는, 속성과 행위를 가지는 주체
- **추상화 (Abstraction)**
  - 객체가 가지는 속성(특징, data-fields/attribute) 및 행위(메소드, methods) 중 필요한 것을 뽑아내어 프로그래밍 코드로 작성하는 것을 의미한다.
- **캡슐화 (Encapsulation)**
  - 객체의 속성과 행위를 하나로 묶음으로써 실제 구현 내용 일부를 외부에 감추어 은닉하는 것을 의미한다.
  - 일반적인 객체지향 언어에서는 접근 제한자(public, package, protected, private) 라는 것이 존재하여 `데이터의 무결성`을 보장한다.
  - 그러나 파이썬에서는 접근 제한자가 없다.
  - 파이썬에서의 캡슐화는 데이터의 무결성을 보장하기 위함으로써 사용되기 보다는 속성과 행위를 하나로 묶어 유지 및 보수를 용이하게 하기 위해 사용한다.
- **상속 (Inheritance)**
  - 상위(부모) 객체로부터 하위(자식) 객체가 특징 및 행위를 물려받는 것을 의미한다.
- **다형성 (Polymorphism)**
  - 하나의 형태가 여러가지 성질을 가질 수 있는 것을 의미한다.
  - 아래의 3가지 개념과 함께 알아두어야 한다.
    - 오버라이딩 (Overriding): 메소드명은 같으나 그 내부에 서로 다른 로직을 가질 수 있도록 `재정의`하는 것을 의미한다.
    - 오버로딩 (Overloading): 메소드명은 같으나 파라미터 순서, 타입, 반환 타입이 다른 것을 의미하며, 이로써 다양한 형태를 갖을 수 있도록 한다.
    - 단, 파이썬은 `기본적`으로 오버로딩을 지원하지 않는다.
    - 연산자 오버로딩 (Operator Overloading): 기존에 정의되어있는 연산자를 다른 방식으로 처리할 수 있도록 정의하는 것을 의미한다.

- **패키지 (Package)**
  - 모듈을 모아놓은 단위
  - 관련된 여러 개의 모듈을 계층적인 디렉터리로 분류해서 저장하고 관리한다.
  - dot(`.`) 연산자를 이용하여 관리할 수 있다.
  - `__init__.py`는 패키지를 인식시켜주는 역할을 수행 -> 특정 디렉토리가 패키지로 인식되기 위해 필요한 파일이다.

- **클래스 (Class)**
  - 새로운 이름 공간을 지원하는 단위: 데이터의 설계도 역할을 한다.
  - 데이터와 데이터를 변경하는 함수(메소드)를 같은 공간 내에 작성한다.
  - 클래스를 정의하는 것은 새로운 자료형을 정의하는 것이고, 인스턴스는 이 자료형의 객체를 생성하는 것이다.
  - 클래스와 인스턴스는 각자의 이름공간을 가지게 되며 유기적인 관계로 연결

용어 정리

용어	설명
클래스 (Class)	class 문으로 정의하며, 속성과 메소드를 가지는 객체
클래스 객체	어떤 클래스를 구체적으로 가리킬 때 사용
인스턴스 (Instance)	'최종적'으로 클래스의 '__init__' 메소드를 호출하여 만들어지는 객체
인스턴스 객체	인스턴스화 된 객체. 파이썬에서는 클래스도 인스턴스 객체로 취급된다.
멤버(변수) 혹은 필드	클래스가 갖는 변수(식별자)
메소드 (Method)	클래스 내에 정의된 함수
속성 (Attribute)	멤버(변수)와 같음
상위(부모) 클래스	기반(base) 클래스. 어떤 클래스의 상위에 있으며 여러 속성을 상속해준다.
하위(자식) 클래스	파생 클래스. 상위 클래스로부터 여러 속성을 상속 받는다.

- **정적 메소드(static method)와 클래스 메소드(class method)**
  - 인스턴스 객체의 멤버에 접근할 필요가 없는 메소드
  - 첫 번째 인자로 인스턴스 객체 참조값을 받지 않는 클래스 내에 정의된 메소드
  - Class 메소드의 첫 번째 인자는 클래스 객체 참조를 위한 객체 참조값
  - `@staticmethod`, `@classmethod` 데코레이터로 손쉽게 구현 가능

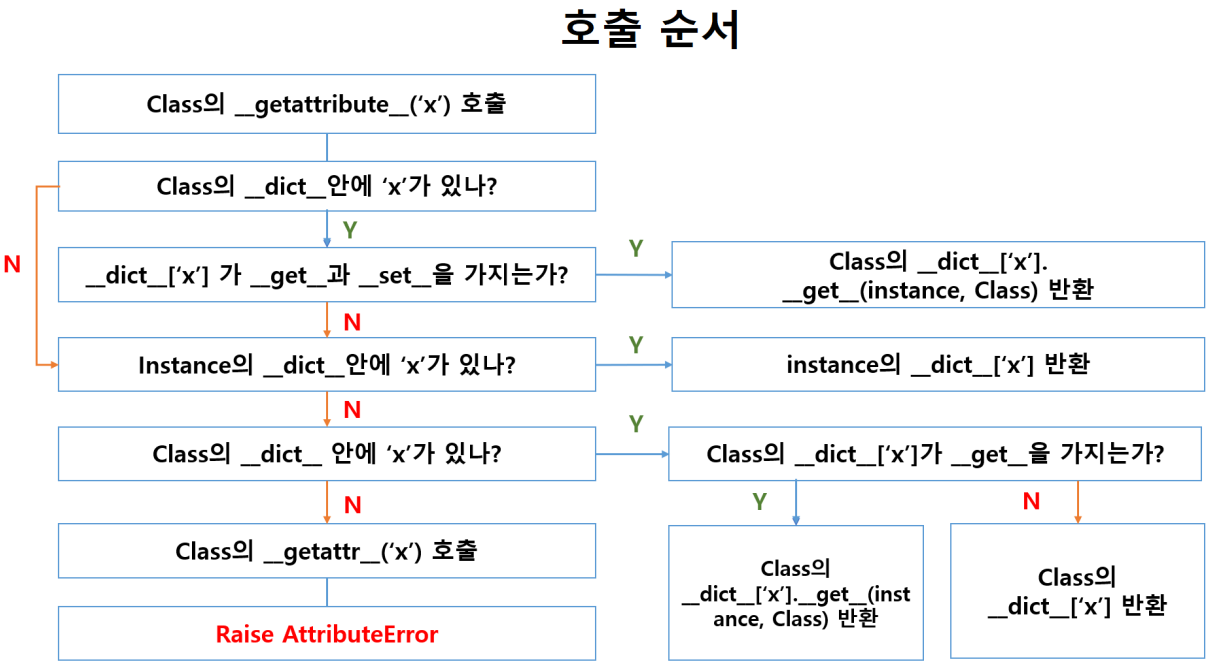
- 클래스 멤버와 인스턴스 멤버

종류	이름공간 (namespace)	공유 범위

클래스 멤버	클래스 이름 공간 내	모든 인스턴스 객체들에 공유
인스턴스 멤버	인스턴스 이름 공간 내	개별 인스턴스 객체에서만 참조

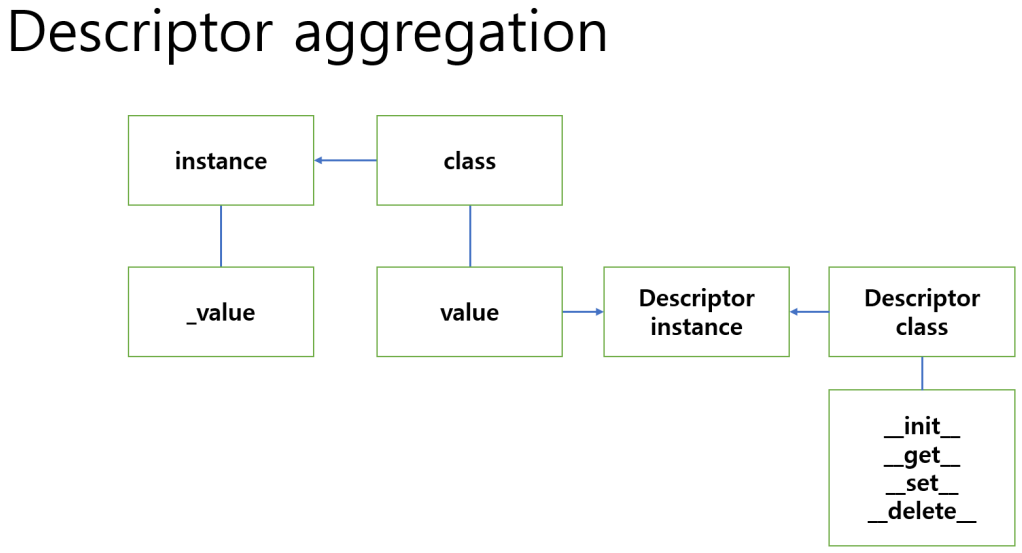
Object Attribute

- 클래스 멤버와 인스턴스 멤버 접근
  - 인스턴스 객체에서 참조하는 멤버의 객체를 찾는 순서는 아래와 같다
    - 인스턴스 멤버
    - 인스턴스 멤버가 없다면 클래스 멤버를 찾음



Descriptor

- Attribute access가 가능하도록 Descriptor protocol의 메소드를 오버라이드하여 `binding-behavior`를 갖는 객체의 attribute
- `__get__`, `__set__`, `__delete__` 3가지 메소드가 오버라이드된 클래스



생성자와 소멸자

- **생성자**: 클래스가 인스턴스화 될 때 실행되는 내용. 일반적으로 초기화 작업이 이루어짐.
  - `__init__` 메소드 내에 작성

    - `__init__` 메소드 내에 작성
- **소멸자**: 클래스 인스턴스가 제거될 때 실행되는 내용.
    - `__del__` 메소드 내에 작성

`__str__` vs `__repr__`

- `__str__` 메소드
    - 객체를 문자열로 반환하는 함수
- `__repr__` 메소드
    - `__str__`과 비슷하지만 "문자열로 객체를 다시 생성할 수 있기 위해" 사용
    - Eval을 수행하면 다시 그 해당 객체가 생성될 수 있어야 한다.

- `__str__` vs `__repr__`

	str	repr
구분	비공식적 문자열 출력	공식적 문자열 출력
목적	사용자가 보기 쉽게	문자열로 객체를 다시 생성할 수 있도록
대상	사용자(End User)	개발자(Developer)

연산자 재정의 (Operator Overloading)

- 연산자에 대해 클래스에 새로운 동작을 정의하는 것
- 파이썬의 클래스는 새로운 데이터 형을 정의하는 것이므로 그에 상응하는 연산자의 재정의가 필요할 수 있다.
- 연산자가 정의되어 있지 않으면 TypeError가 발생
- 파이썬에서는 사용하는 거의 모든 연산에 대해 새롭게 정의할 수 있다.
    - 수치 연산자 오버로딩
    - 역이항 연산자 오버로딩
    - 확장 산술 연산자 오버로딩
    - 비교 연산자 오버로딩

연산자	수치 연산자 메소드	역이행 연산자 메소드
+	__add__	__radd__
-	__sub__	__rsub__
*	__mul__	__rmul__
/	__truediv__	
//	__floormod__	__rfloormod__
%	__mod__	__rmod__
divmod()	__divmod__	__rdivmod__
pow(), **	__pwd__	__rpow__
<<	__lshift__	__rlshift__
>>	__rshift__	__rrshift__
&	__and__	__rand__
^	__xor__	__rxor__
|	__or__	__ror__

연산자	확장 산술 연산자 오버로딩
+=	__iadd__
-=	__isub__
*=	__imul__
//=	__ifloormod__
/=	__idiv__
%=	__imod__
**=	__ipow__
<<=	__ilshift__
>>=	__irshift__
&=	__iand__
^=	__ixor__
|=	__ior__

연산자	비교 연산자 오버로딩
<	__lt__
<=	__le__
>	__gt__
>=	__ge__
==	__eq__
!=	__ne__

Python 소스 코드로 살펴보기

In [1]:
```
# class 정의
class A: # 클래스 명은 관례상 camel case로 작성한다.
         # (python에서는 pascal case도 camel case라고 부른다.)

    # class 변수 또는 class attribute로 불린다.
    x = 3

    # (instance) method로 불린다.
    def y(self): # method의 parameter에 self는 instance되는 객체 자신을 의미한다.
                 # (관례상 self 혹은 cls라는 키워드를 많이 사용한다.)

        # instance 변수 또는 instance attribute로 불린다.
        self.t = 1
```

In [2]:
```
# instance 객체 A 생성
ttt = A()
```

In [3]:
```
ttt.x
```

Out[3]: 3

In [4]:
```
# y method를 호출하면 t라는 instance 변수를 호출할 수 있다.
ttt.y()
```

In [5]:
```
ttt.t
```

Out[5]: 1

python의 기본 자료형들은 모두 객체다.

In [6]:
```
# constructor 방식으로 int형 자료를 선언하여 instance 객체 생성
a = int()
```

In [7]:
```
a
```

Out[7]: 0

dir 함수는 객체의 attribute와 method를 확인할 수 있는 함수이다.

In [8]:
```
dir(a)
```

Out[8]:
```
['__abs__',
 '__add__',
 '__and__',
 '__bool__',
 '__ceil__',
 '__class__',
 '__delattr__',
 '__dir__',
 '__divmod__',
 '__doc__',
 '__eq__',
 '__float__',
 '__floor__',
 '__floordiv__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getnewargs__',
 '__gt__',
 '__hash__',
 '__index__',
 '__init__',
 '__init_subclass__',
 '__int__',
 '__invert__',
 '__le__',
 '__lshift__',
 '__lt__',
 '__mod__',
 '__mul__',
 '__ne__',
 '__neg__',
 '__new__',
 '__or__',
 '__pos__',
 '__pow__',
 '__radd__',
 '__rand__',
 '__rdivmod__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rfloordiv__',
 '__rlshift__',
 '__rmod__',
 '__rmul__',
 '__ror__',
 '__round__',
 '__rpow__',
 '__rrshift__',
```

```
'__rshift__',
'__rsub__',
'__rtruediv__',
'__rxor__',
'__setattr__',
'__sizeof__',
'__str__',
'__sub__',
'__subclasshook__',
'__truediv__',
'__trunc__',
'__xor__',
'bit_length',
'conjugate',
'denominator',
'from_bytes',
'imag',
'numerator',
'real',
'to_bytes']
```

In [9]:
```
class X:
    a = 3

    # 생성자(constructor)
    def __init__(self):
        self.a = 7

    def set_b(self, b):
        self.b = b
```

In [10]:
```
X() # <-- 생성자 호출
```

Out[10]:
```
<__main__.X at 0x7fc7a84bb080>
```

In [11]:
```
t = X()
t.set_b(3)
t.b
```

Out[11]:
```
3
```

In [12]:
```
s = X()
s.set_b(10)
s.b
```

Out[12]:
```
10
```

In [13]:
```
print('t.a:', t.a)
print('s.a:', s.a)
print('X.a:', X.a)
```

```
t.a: 7
s.a: 7
X.a: 3
```

In [14]:
```
class Y:
    a = 3

    def __init__(self, b):
        self.a = b

    def ttt(self):
        print(self.a)
```

In [15]:
```
a = Y(4)
a.ttt()
Y.ttt(a) # 위의 a.ttt()와 같은 의미다.
```

```
4
4
```

In [16]:
```
class Y:
    a = 3

    def __init__(self, b):
        self.a = b

    # class method를 선언할 때는 아래와 같이 decorator를 붙여준다.
    @classmethod
    def ttt(self):
        print(self.a)
```

In [17]:
```
a = Y(4)
a.ttt()
# class method로 선언된 method는 self라는 parameter의 의미가 달라진다.
Y.ttt(a)
```

```
3

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-5809c18c7564> in <module>
      2 a.ttt()
      3 # class method로 선언된 method는 self라는 parameter의 의미가 달라진다.
----> 4 Y.ttt(a)

TypeError: ttt() takes 1 positional argument but 2 were given
```

class method 첫 번째 parameter `self` 는 instance 객체가 아닌 class를 의미한다.

따라서 class method인 ttt를 호출하면 class attribute를 출력한다.

In [18]:
```
Y.ttt()
```

```
3
```

`__` 가 붙은 method는 `magic method` 이다.

보통 내부 구조를 구현하기 위해 사용되는 키워드이다.

In [19]:
```
# vars는 객체의 attribute들을 dict로 반환시켜주는 함수이다.
vars(a)
```

Out[19]:
```
{'a': 4}
```

In [20]:
```
# magic method
a.__dict__
```

Out[20]:
```
{'a': 4}
```

In [21]:
```
a.__class__
```

Out[21]:
```
__main__.Y
```

In [22]:
```
type(a)
```

Out[22]:
```
__main__.Y
```

In [23]:
```
a.__dir__()
```

Out[23]:
```
['a',
 '__module__',
 '__init__',
 'ttt',
 '__dict__',
 '__weakref__',
 '__doc__',
 '__repr__',
 '__hash__',
 '__str__',
 '__getattribute__',
 '__setattr__',
 '__delattr__',
 '__lt__',
 '__le__',
 '__eq__',
 '__ne__',
 '__gt__',
 '__ge__',
 '__new__',
 '__reduce_ex__',
 '__reduce__',
 '__subclasshook__',
 '__init_subclass__',
 '__format__',
 '__sizeof__',
 '__dir__',
 '__class__']
```

In [24]:
```
dir(a) # __dir__()의 결과를 정렬하여 보여준다.
```

Out[24]:
```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'a',
 'ttt']
```

In [25]:
```
class Y:
    # Doc-string
    # class를 설명하기 위해서 주석과 같은 의미로써 사용한다.
```

```
    '''클래스 Y입니다.'''
    a = 3

    def __init__(self, b):
        self.a = b

    @classmethod
    def ttt(self):
        print(self.a)
```

In [26]:
```
Y.__doc__
```

Out[26]: '클래스 Y입니다.'

In [27]:
```
# shift-tab 키를 눌러 확인할 때 설명이 나오는 부분이 바로 Docstring이다.
??Y
```

In [28]:
```
# Y라는 클래스가 선언된 모듈명을 의미한다.
Y.__module__
```

Out[28]: '__main__'