



파이썬기초

6주차 - 2교시
객체지향의 필요성





학습내용

- 클래스를 사용하는 이유
- 파이썬에서의 객체



학습목표

- 파이썬에서 객체를 사용해야 하는 상황과 효과를 확인해보고 프로그래밍할 수 있다.
- 파이썬에서 사용하는 타입을 객체로 사용하는 것을 확인하고 이용하여 프로그래밍 할 수 있다.



생각해 봅시다

클래스를 이용하여 구조화하여
프로그래밍을 어떻게 할지를
생각해 봅시다.



01



클래스를 사용하는 이유



1 | 클래스를 사용하는 이유

01 클래스를 사용하는 이유

- 자신의 코드를 **다른 사람이 손쉽게 사용**할 수 있도록 설계하기 위함
 - 코드를 좀 더 **손쉽게 선언**할 수 있음
- 단순히 이차원 리스트로 선언할 수 있는 것을 객체 지향 프로그래밍의 개념을 적용시킴으로 좀 더 **명확하게** 저장된 데이터를 확인할 수 있음

01 클래스를 사용하는 이유

1 | 클래스를 사용하는 이유

02 2차원 리스트 사용 예제

```
In [1]: # 이차원 리스트
players = [['Messi', '바르셀로나', 10], ['Ramos', '마드리드', 4],
            ['Park', '맨체스터', 13], ['SON', '토트넘', 1]]
print(players)

for p in players:
    print('{0} {1} {2}'.format(p[0], p[1], p[2]))
```

01 클래스를 사용하는 이유

1 | 클래스를 사용하는 이유

02 2차원 리스트 사용 예제

결과

```
[['Messi', '바르셀로나', 10], ['Ramos', '마드리드', 4], ['Park', '맨체스터', 13],  
['SON', '토트넘', 1]]  
Messi 바르셀로나 10  
Ramos 마드리드 4  
Park 맨체스터 13  
SON 토트넘 1
```

1 | 클래스를 사용하는 이유

03 객체를 이용하는 예제

```
In [8]: new_players = []
```

```
In [9]: new_players.append(FootballPlayer("Messi", "바르셀로나", 10))  
new_players.append(FootballPlayer("Ramos", "마드리드", 4))  
new_players.append(FootballPlayer("Park", "맨체스터", 13))  
new_players.append(FootballPlayer("SON", "토트넘", 1))
```

```
In [10]: for fp in new_players:  
         print(fp)
```


01 클래스를 사용하는 이유

1 | 클래스를 사용하는 이유

03 객체를 이용하는 예제

결과

Messi (팀:바르셀로나, 등번호:10)

Ramos (팀:마드리드, 등번호:4)

Park (팀:맨체스터, 등번호:13)

SON (팀:토트넘, 등번호:1)

02



파이썬의 객체



1 | 파이썬에서의 객체

01 객체

- 파이썬은 모든 것을 **객체**로 취급함

int

float

str

→ 모두 객체로서, 이 객체도 속성과 기능으로 구성됨

02 파이썬에서의 객체

1 | 파이썬에서의 객체

02 타입으로 확인

```
In [13]: a=10  
         b=3.14  
         c='str'  
         print(type(a))  
         print(type(b))  
         print(type(c))
```

결과

```
<class 'int'>  
<class 'float'>  
<class 'str'>
```

1 | 파이썬에서의 객체

02 타입으로 확인

```
In [14]: d=[1,2,3]
          e=(1,2,3)
          f={1:2, 3:4, 4:5}
          g={1,2,3}
          print(type(d))
          print(type(e))
          print(type(f))
          print(type(g))
```

결과

```
<class 'list'>
<class 'tuple'>
<class 'dict'>
<class 'set'>
```

2 | 객체의 속성과 메소드 확인

01 객체의 속성과 메소드 확인

```
In [17]: a=5  
print(dir(a))
```

결과

```
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__',  
 '__dir__', '__divmod__', '__doc__', '__eq__', '__float__', '__floor__', '__floordiv__',  
 '__format__', '__ge__', '__getattr__', '__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__', 'as_integer_ratio', 'bit_count', 'bit_length', 'conjugate',  
 'denominator', 'from_bytes', 'imag', 'numerator', 'real', 'to_bytes']
```

2 | 객체의 속성과 메소드 확인

01 객체의 속성과 메소드 확인

- 예제

In [18]: # 5(101) 비트 표현했을 때 길이
a.bit_length()

결과

3

2 | 객체의 속성과 메소드 확인

01 객체의 속성과 메소드 확인

- 리스트 타입 확인

In [23]: `print(dir(d))`

결과

```
['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__',  
 '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',  
 '__getattr__', '__getitem__', '__gt__', '__hash__', '__iadd__',  
 '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__',  
 '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',  
 '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__',  
 '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count',  
 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```


2 | 객체의 속성과 메소드 확인

01 객체의 속성과 메소드 확인

- 문자열 타입 확인

In [24] : `print(dir(c))`

결과

```
['_add_', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__',  
 '__format__', '__ge__', '__getattr__', '__getitem__', '__getnewargs__', '__gt__',  
 '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__',  
 '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',  
 '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',  
 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find',  
 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit',  
 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join',  
 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'removeprefix', 'removesuffix', 'replace',  
 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith',  
 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

2 | 객체의 속성과 메소드 확인

02 클래스의 특별한 메소드

- `__del__()` 메소드

➤ 소멸자(Destructor), 생성자와 반대로 인스턴스 삭제할 때 자동 호출

→ 객체를 제거할 때 `del(객체)` 함수를 이용하는데 이때 호출됨

2 | 객체의 속성과 메소드 확인

02 클래스의 특별한 메소드

- `__str__()` 메소드

➤ `str()` 함수 호출할 때 `__str__()` 함수가 자동으로 호출

2 | 객체의 속성과 메소드 확인

02 클래스의 특별한 메소드

- `__add__()` 메소드

➤ 인스턴스 사이에 덧셈 작업이 일어날 때 실행되는 메소드

→ 일반적으로 덧셈(+) 연산은 숫자나 문자열 등에만 작동하지만 **인스턴스 사이의 덧셈 작업이 가능함**

2 | 객체의 속성과 메소드 확인

02 클래스의 특별한 메소드

- 비교 메소드

➤ 인스턴스 사이의 비교 연산자(<, <=, >, >=, ==, != 등) 사용할 때 호출

예 `__lt__()`, `__le__()`, `__gt__()`, `__ge__()`,
`__eq__()`, `__ne__()`

3 | 클래스 관련 함수

03 isinstance() 함수

isinstance() 함수

상속 관계에 따라서 객체가 어떤 클래스를 기반으로 만들었는지 확인할 수 있게 해주는 함수

 **isinstance**
인스턴스, 클래스

3 | 클래스 관련 함수

03 isinstance() 함수

- 예제

```
In [25]: # 클래스 정의
class Player :
    def __init__(self):
        pass
```

```
In [26]: # 플레이어를 생성
player = Player()
```

3 | 클래스 관련 함수

03 isinstance() 함수

- 예제

In [27]: `print('isinstance(player, Player)', isinstance(player, Player))`

결과 `isinstance(player, Player) True`



파이썬기초

NEXT
객체지향 활용

