

Day40; 20221102



GitHub - u8yes/Python

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

<https://github.com/u8yes/Python>

u8yes/Python

1 Contributor 0 Issues 2 Stars 0 Forks

fieldinheritance.py

```
# 데이터 속성(field) 상속

"""
class A:
    def __init__(self): # '__init__'는 필드값을 초기화하라고 정의해준 것.
        print("A.__init__() 생성자 호출")
        self.message = "Heavenly"

class B(A):
    def __init__(self):
        print("B.__init__() 생성자 호출")

if __name__ == '__main__':
    obj = B() # B를 호출하면서 A의 생성자를 호출하면 필드를 초기화할 의무를 가지고 있다.(상속)

    print(obj.message) # AttributeError: 'B' object has no attribute 'message'
```

```

"""
class A:
    def __init__(self): # '__init__'는 필드값을 초기화하라고 정의해준 것.
        print("A.__init__() 생성자 호출")
        self.message = "Heavenly"

class B(A):
    def __init__(self):
        A.__init__(self)           # 생성자 호출을 명시적으로 표현.
        print("B.__init__() 생성자 호출")

if __name__ == '__main__':
    obj = B() # B를 호출하면서 A의 생성자를 호출하면 필드를 초기화할 의무를 가지고 있다.(상속)
    print(obj.message)

```

A.**init()** 생성자 호출

B.**init()** 생성자 호출

Heavenly

super.py

```

class A:
    def __init__(self):
        print("A.__init__()")
        self.message = "Heavenly"

class B(A):
    def __init__(self):
        super().__init__()
        print("B.__init__()")

    print("self.message is " + self.message)

if __name__ == '__main__':
    obj = B()

    print("__main__"+obj.message)

#####
class Base:
    def __init__(self):
        print("Base")

class Derived(Base):
    pass
    # def __init__(self):
    #     super().__init__()

print("=====")

d = Derived()

```

A.**init()**

B.**init()**

self.message is Heavenly

__main__Heavenly

Base

multiinheritance.py

```
# 다중 상속 # UML이라는 단어는 상속 모양을 만들 수 있다는 것.

class A:
    def method(self):
        print("A")

class B(A):
    def method(self):
        print("B")

class C(A):
    def method(self):
        print("C")

class D(B,C):
    pass

if __name__ == '__main__':
    obj = D()
    obj.method() # B가 호출됨.
```

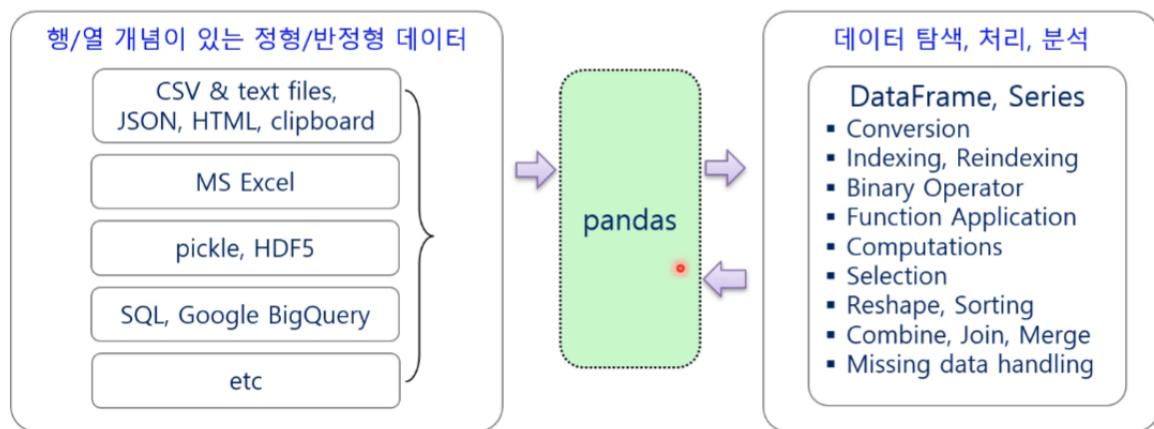
결과: B

속성 ← 멤버변수(field)

기능 ← 멤버 메서드

pandas 특징

- ▣ 다양한 형태의 file, 생성자를 사용하여 DataFrame을 만든다
- ▣ DataFrame, Series, Index 등의 객체를 사용하여 데이터 처리 및 분석을 수행 한다
 - 위의 객체들은 ndarray를 기반으로 데이터의 처리, 분석을 효율적으로 할 수 있다



변수 선언 때 참조자료형은 Stack에 저장, new 하는 것은 Heap영역에 저장.

상속에서만 instanceof 사용함. 할당된 object가 자식을 재정의 된 것으로 바라보는지 heap영역에서 체크함.

SQLD)

단일 행 함수

- 추출되는 각 행마다 작업을 수행
- 각 행마다 하나의 결과를 반환
- SELECT, WHERE, ORDER BY, UPDATE의 SET 절에 사용 가능
- 데이터 타입 변경 가능
- 중첩해서 사용 가능

연산	/	일수
DATE + NUMBER		
DATE - NUMBER	/	시간
DATE + NUMBER/24		
DATE1 - DATE2		

ORACLE은
세기, 년, 월, 일, 시, 분, 초
형식의 날짜를 내부적으로는
숫자 형태로 저장.

- 1일 SYSDATE + 1
- 1시간 SYSDATE + 1/24
- 10분 SYSDATE + 1/24/6
- 1분 SYSDATE + 1/24/60
- 매일 밤 11시 TRUNCATE(SYSDATE) + 23/24

CASE WHEN LOC = 'NEW YORK' THEN 'EAST'

[SIMPLE_CASE_EXPRESSION 문장]

```
SELECT LOC,  
CASE LOC WHEN 'NEW YORK' THEN 'EAST'  
ELSE 'ETC'  
END as AREA  
FROM DEPT;
```

46
P63

사원 테이블에서 MGR의 값이 7698과 같으면 NULL을 표시하고, 같지 않으면 MGR을 표시하려고 한다. 빈칸에 들어갈 함수는?

```
SELECT ENAME, EMPNO, MGR,  
NULLIF(MGR, 7689) as NVL  
FROM EMP;
```

```
override("ABC") # .method가 없다고 반응해줌 # AttributeError: 'str' object has no attribute 'method'  
D:/heaven_dev/workspaces/Python/src(python)/chap08_20221101/06_overriding1.py  
def override(overriding: {method}) -> None  
::
```

overriding1.py

```
# Overriding  
  
class A:  
    def method(self):  
        print("A")  
  
class B(A):  
    def method(self):  
        print("B")
```

```

class C(A):
    def method(self):
        print("C")

# ---- 여기까지는 다중상속이 아니다. ----

def override(overriding):
    overriding.method()

if __name__ == '__main__':
    a = A()
    # a.method()

    b = B()
    # b.method()

    c = C()
    # c.method()

    override(a)
    override(b)
    override(c)

# override("ABC") # .method가 없다고 반응해줌 # AttributeError: 'str' object has no attribute 'method'

```

A
B
C

overriding2.py

```

class Car:
    def ride(self):
        print("Run")

class FlyingCar(Car):
    def ride(self):
        super().ride()
        # 부모의 생성자를 호출할 때는 super() 괄호를 붙여줘야한다. super(x) / super()(o)
        print("FLY")

if __name__ == "__main__":
    # car = Car()
    # car.ride()

    my_car = FlyingCar()
    my_car.ride()

```

Run
FLY

callable

미국·영국 [kə'leɪbl] ◎

(형용사)

청구하는 대로 지급되는, 부를 수 있는, <채권 등이> 만기 전[수시] 상환의

[영어사전 결과 더보기](#)

decorator.py

```
# 데코레이터(Decorator)

class Callable:
    def __init__(self):
        pass

    def __call__(self):
        print("I am called")

if __name__ == '__main__':
    obj = Callable()

    obj() # I am called
    # 인스턴스 뒤에 괄호()를 붙여 호출하면, 내부적으로는 __call__(self) 메서드가 호출된다.
    # 데코레이터(Decorator)
```

sql))

0과 NULL을 포함한 나눗셈

```
SELECT SAL/COMM FROM EMP_Q WHERE ENAME = 'KING';
SELECT SAL/COMM FROM EMP_Q WHERE ENAME = 'FORD';
SELECT SAL/COMM FROM EMP_Q WHERE ENAME = 'SCOTT';
```

EMP_Q

숫자/0 → error
숫자/NULL → NULL

ENAME (문자타입)	SAL (숫자타입)	COMM (숫자타입)
KING	0	300
FORD	5000	0
SCOTT	1000	

- ① 0, NULL, NULL
- ② 0, error 발생, error 발생
- ③ error 발생, error 발생, NULL
- ✓ ④ 0, error 발생, NULL

48
P64

다음과 같은 데이터 상황에서 SQL의 수행 결과로
가장 적절할 것은?

TAB1

C1	C2	C3
1	2	3
2		3
	3	

**SELECT SUM(COALESCE
(C1, C2, C3))
FROM TAB1;**

- ① 0 ② 1
③ 6 ④ 14

COALESCE(expr1, expr2, ...)
처음으로 널이 아닌 것을 리턴

decorator_constructor.py

```
# 데코레이터 사용 용도(생성자=constructor)

class MyDecorator:
    def __init__(self, f): # self에 주소값을 넣을 것이다.
        print("initializing MyDecorator...")
        self.func = f # func라는 field 정의

    def __call__(self):
        print(f"Begin:{self.func.__name__}")

        self.func() # __call__() 메서드가 호출되면 생성자에서 저장해둔 함수(데이터 속성)를 호출

        print(f"End:{self.func.__name__}")

    def print_hello():
        print("Hello.")

if __name__ == "__main__":
    hello = MyDecorator(print_hello) # f에 print_hello주소값을 저장
        # MyDecorator의 인스턴스가 만들어지며, __init__() 메서드가 호출
        # print_hello 식별자는 앞에서 정의한 함수가 아닌 MyDecorator의 객체.
    hello() # __call__(self) 메서드 덕에 MyDecorator 객체를 호출하듯 사용할 수 있음.
```

initializing MyDecorator...
Begin:print_hello
Hello.
End:print_hello

Decorator_@.py

```
# @Decorator

class MyDecorator:
    def __init__(self, f): # self에 주소값을 넣을 것이다.
        print("initializing MyDecorator...")
        self.func = f # func라는 field 정의

    def __call__(self):
        print(f"Begin:{self.func.__name__}")

        self.func() # __call__() 메서드가 호출되면 생성자에서 저장해둔 함수(데이터 속성)를 호출

        print(f"End:{self.func.__name__}")

if __name__ == "__main__":
    @MyDecorator # 데코레이터가 __call__(self)가 호출되게 만들어줌.
    def print_hello():
        print("Hello.")

    print_hello()
```

initializing MyDecorator...
Begin:print_hello
Hello.
End:print_hello

iterator.py

```
# Iterator와 순회 가능한 객체

for i in range(5): # range(0, 5) # 0,1,2,3,4
    print(i)

print("-----")

iterator = range(3).__iter__()
print(iterator.__next__())
print(iterator.__next__())
print(iterator.__next__())

# print(iterator.__next__()) # error

class MyRange: # range() 함수와 같은 일을 하는 클래스 정의.
    def __init__(self, start, end):
```

```

        self.current = start
        self.end = end

    def __iter__(self): # 한번만 반환해주고 for문에 포함안 됨.
        print("iter")
        return self

    def __next__(self): # 계속 반복해서 for문이 거치게 됨.
        print("next")
        if self.current < self.end:
            current = self.current
            self.current += 1
            return current
        else:
            raise StopIteration # raise는 자바에서의 throw기능과 같다, 예외를 던져주는 것.

    if __name__ == '__main__':
        print("-----")
        for i in MyRange(0, 5):
            print(i)

```

0
1
2
3
4

0
1
2

iter
next
0
next
1
next
2
next
3
next
4
next

sql)

51
P66

어느 기업의 직원 테이블(EMP)이 직급(GRADE)별로 사원 500명, 대리 100명, 과장 30명, 차장 10명, 부장 5명, 직급이 정해지지 않은(NULL) 사람 25명으로 구성되어 있을 때, SQL1부터 SQL3까지 순차적으로 실행한 결과 건수를 순서대로 나열한 것으로 가장 적절한 것은?

SQL1) SELECT COUNT(GRADE) FROM EMP;
SQL2) SELECT GRADE FROM EMP
WHERE GRADE IN ('차장', '부장', '널');
SQL3) SELECT GRADE, COUNT(*)
FROM EMP
GROUP BY GRADE;

사원:	500명	
대리:	100명	670, 15, 5
과장:	30명	645, 15, 6
차장:	10명	670, 40, 6
부장:	5명	670, 40, 6
Null:	25명	670, 40, 6

generator.py

```
def generator():
    yield 0
    yield 1
    yield 2
    yield 3

iterator = generator()
print(iterator.__next__())
print(iterator.__next__())
print(iterator.__next__())
print(iterator.__next__())
# print(iterator.__next__()) # error

#####
#####
```



```
def yourRange(start, end):
    current = start

    while current < end:
        yield current
        current += 1

if __name__ == '__main__':
    print("====")
    for i in yourRange(0, 4):
        print(i)
```

0
1
2
3

0
1
2
3

Exception.py

```
def my_power(y):
    print("숫자를 입력하세요 : ")
    x = int(input())
    return x ** y

if __name__ == '__main__':
    print(my_power(2))
```

숫자를 입력하세요 : 9

81
