(1번)

```
class Base:
    def base_method(self):
        print("base_method")

class Derived(Base):
    pass

base = Base();
base.base_method()

derived = Derived()
derived.base_method()
```

(2번)

```
class Base:
    def base_method(self):
        print("base_method")

class Derived:
    def __init__(self):
        self.instanceBase = Base()

    def base_method(self):
        self.instanceBase.base_method()

base = Base();
base.base_method()

derived = Derived()
derived.base_method()
```

(3번)

```
class Logger:
    def WriteOnFile(self):
        print("파일에 로그를 남깁니다.")

class NetworkLogger(Logger):
    def WriteOnNetwork(self):
        print("네트워크에 로그를 남깁니다.")


logger = NetworkLogger()
logger.WriteOnFile()
logger.WriteOnNetwork()
```

(4번)

```
class ArmorSuite:
    def armor(self):
        print("armored")

class IronMain(ArmorSuite):
    pass

def get_armored(suite):
    suite.armor()

suite = ArmorSuite()
get_armored(suite)

iron_man = IronMain()
get_armored(suite)
```

(5번)

```
class A:
    def __init__(self):
        print("A.__init__()")
        self.message = 'hello'

class B(A):
    def __init__(self):
        print('B.__init__()')

obj = B()
print(obj.message)
```

```
class A:
    def __init__(self):
        print("A.__init__()")
        self.message = 'hello'

class B(A):
    def __init__(self):
        A.__init__(self)
        print('B.__init__()')

obj = B()
print(obj.message)
```

```
class A:
    def __init__(self):
        print("A.__init__()")
        self.message = 'hello'

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

obj = B()
print(obj.message)
```

(6 번)

```
class A:
    pass
class B:
    pass
class C:
    pass
class D(A, B, C):
    pass
```

```
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

obj = D()
obj.method()
```

(7 번)

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

class FlyingCar(Car):
    def ride(self):
        super().ride()
        print("Fly")

my_car = FlyingCar()
my_car.ride()
```

(8 번)

```
class Callable :
    def __call__(self):
        print('I am calles')

obj = Callable()
obj()
```

(9 번)

```
class MyDecorator:
    def __init__(self, f):
        print("Init : MyDecorator")
        self.func = f

    def __call__(self):
        print("start", self.func.__name__)
        self.func()
        print("end", self.func.__name__)

def print_hello():
    print("hello")

print_hello = MyDecorator(print_hello)
print_hello()
```

(10 번)

```
class MyDecorator:
    def __init__(self, f):
        print("Init : MyDecorator")
        self.func = f

    def __call__(self):
        print("start", self.func.__name__)
        self.func()
        print("end", self.func.__name__)

@MyDecorator
def print_hello():
    print("hello")

@MyDecorator
def print_bye():
    print("byte")

print_hello()
print("*" * 30)
print_bye()
```

(11 번)

```
iterator = range(3).__iter__()

print("#1")
print(type(iterator))

print("#2")
print(iterator.__next__())
print(iterator.__next__())
print(iterator.__next__())
print(iterator.__next__()) #에러발생
```

(12 번)

```
class MyRange :
    def __init__(self, start, end) :
        self.current = start
        self.end = end

    def __iter__(self):
        return self

    def __next__(self):
        if self.current < self.end :
            current = self.current
            self.current += 1
            return current
        else :
            raise StopIteration()

for i in MyRange(0,5) :
    print(i)
```

(13 번)

```
def YourRange(start, end) :
    current = start
    while current < end :
        yield current
        current+= 1
    return

for i in YourRange(0,5) :
    print(i)
```

(14 번)

```
from abc import ABCMeta
from abc import abstractmethod

class MetaDuck(metaclass=ABCMeta) :
    @abstractmethod
    def Quack(self):
        pass

class Duck(MetaDuck) :
    pass

duck = Duck()
```

```
from abc import ABCMeta
from abc import abstractmethod

class MetaDuck(metaclass=ABCMeta) :
    @abstractmethod
    def Quack(self):
        pass

class Duck(MetaDuck) :
    def Quack(self) :
        print("[Duck] Quack")

Duck().Quack()
```