1. Lookup Hierarchy Order
-> 자식 클래스 -> 부모 클래스 순서로 찾는다.

ex)

```
class Animal( ):
    def __init__(self, name):
        self.name = name

class Dog(Animal):
    def fetch(self, thing):
        print("%s goes after the %s!" %(self.name, thing))

r = Dog('Rover')
```

-> Dog에는 __init__이 없으므로 Animal 생성자가 호출된다.
-> 이 때 __init__(self, name)에서 self는 Dog를 가리킨다.

2. Method Resolution Order(MRO)

ex1) Python은 기본적으로 depth 먼저 탐색한다.

```
class A():
    def do_this():

class B(A):
    pass

class C():
    def do_this();

class D(B,C):
        pass

d_instance = D()
d_instance.dothis()
```

- D -> B -> A -> C로 탐색한다.

ex2) Diamond shape의 경우에는 자식 먼저 탐색한다.

```
class A():
    def dothis()

class B(A):
    pass

class C(A);
    def dothis()

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

- D -> B -> C -> A로 탐색한다.

3. Inheriting the constructor

1) __init__ 도 동일하게 상속이 된다.
2) 만약 자식 클래스에 __init__이 없으면 부모 클래스에서 찾는다.
3) super( ) 함수로서 부모 클래스 생성자를 부를 수 있다.

ex)

```
class Dog(Animal):
    def __init__(self, name):
        super(Dog, self).__init__(name)
        self.breed = 'A'
```

참고) built_in class 확장

ex1) dictionary 상속

```
class myDict(dict):
    def __setitem__(self, key, val:
        print("setting a key and value!")
        dict.__setitem__(self, key, value)
```

ex2) list 상속

```
class MyList(list):
    def __getitem__(self, index):
        if index == 0: raise IndexError
        if index  > 0: index = index - 1
        return list.__getitem__(self, index)

    def __setitem__(self, index, value):
        if index == 0: raise IndexError
        if index > 0: index = index - 1
        list.__getitem__(self, index, value)
```

4. Abstract Class

```
import abc

class GetterSetter( ):
    __metaclass__ = abc.ABCMeta

    @abc.abstractmethod
    def set_val(self, input):
        return

    @abc.abstractmethod
    def get_val(self);
        return

class MyClass(GetterSetter):
    def set_val(self, input):
        self.value = input

    def get_val(self):
        return self.value
```

5. Inheritance Method 종류
    - Inherit: simply use the parent class' defined method.
    - override / overload: provide child's own version of a method.
    - extend: do work in addition to that in parent's method
    - provide: implement abstract method that parent requires.

ex) method overriding 예시

```
class GetSetParent():
    def set_val(self, value):
        self.val = value

class GetSetInt(GetSetParent):
    def set_val(self, value):
        if not instance(value, int):
            value = 0
        super(GetSetInt, self).set_val(value)    # specializing

class GetSetList(GetSetParent):
    def set_val(self, value)
        self.vallist.append(value)
```