```
>>> class Value:
...     def __init__(self, arg):
...             self.arg = arg
...     def get_arg(self):
...             return self.arg
...
>>> boo = Value(1)
>>> boo.get_arg()
1
>>> del Value
>>> boo.arg
1
>>> boo.get_arg()
1
>>> boo2 = Value(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Value' is not defined. Did you mean: 'False'?
# Previously created objects are not affected by delete operation
>>> class Value:
...     MAX_VALUE = 1000
...     def __init__(self, arg):
...             self.arg = arg
...     def get_arg(self):
...             return self.arg
...     def is_greater(self):
...             return self.arg > self.MAX_VALUE
...
>>> boo = Value(1)
>>> boo.get_arg()
1
>>> boo.is_greater()
False
>>> del Value
>>> boo.get_arg()
1
>>> boo.is_greater()
False
>>> boo.arg = 100000
>>> boo.is_greater()
True
>>>
# Class attributes are not affected by the delete operator
```

```
>>> class Value:
...     def __init__(self, arg):
...             self.arg = arg
...     def get_arg(self):
...             return self.arg
...
>>> class AnotherValue(Value):
...     def __init__(self, arg1, arg2):
...             super().__init__(arg1)
...             self.arg2 = arg2
...     def get_arg2(self):
...             return self.arg2
…
>>> del Value
>>> boo = AnotherValue(1,2)
>>> boo.get_arg()
1
>>> boo.get_arg2()
2
>>> del AnotherValue
>>> boo2 = AnotherValue(1, 2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'AnotherValue' is not defined
# Deleting a class that specifies a base class does not affect previously created objects, but
prevents you from creating new objects
>>> class Value:
...     MAX_VALUE = 1000
...     def __init__(self, arg) :
...             self.arg = arg
...     def get_arg(self):
...             return self.arg
...     def is_greater(self):
...             return self.arg > self.MAX_VALUE
...
>>> class AnotherValue(Value):
...     OTHER_MAX_VALUE = 10000
...     def __init__(self, arg1, arg2):
...             super().__init__(arg1)
...             self.arg2 = arg2
...     def get_arg2(self):
...             return self.arg2
...     def is_greater_2(self):
```

```
...             return self.arg2 > self.OTHER_MAX_VALUE
...
>>> boo = AnotherValue(1, 2)
>>> boo.is_greater()
False
>>> boo.is_greater_2()
False
>>> del Value
>>> boo.is_greater()
False
>>> boo.is_greater_2()
False
>>> del AnotherValue
>>> boo.is_greater()
False
>>> boo.is_greater_2()
False
# Class attributes are not affected by deleting a class with a base class
>>> class Value:
...     MAX_VALUE = 1000
...     def __init__(self, arg):
...             self.arg = arg
...     def get_arg(self):
...             return self.arg
...     def is_greater(self):
...             return self.arg > self.MAX_VALUE
...
>>> class AnotherValue:
...     OTHER_MAX_VALUE = 10000
...     def __init__(self, arg1, arg2):
...             super().__init__(arg1)
...             self.arg2 = arg2
...     def get_arg2(self):
...             return self.arg2
...     def is_greater_2(self):
...             return self.arg2 > OTHER_MAX_VALUE
...
>>> del Value
>>> boo = AnotherValue()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: AnotherValue.__init__() missing 2 required positional arguments: 'arg1' and 'arg2'
>>> boo = AnotherValue(1,2)
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 4, in __init__
TypeError: object.__init__() takes exactly one argument (the instance to initialize)
>>>
# deleting the base class of a derived class prevents the creation of new derived classes

>>> class Value:
...     def get_ones(self):
...             return 1
...
>>> class AnotherValue(Value):
...     pass
...
>>> boo = AnotherValue()
>>> boo.get_ones()
1
>>> del Value
>>> boo.get_ones()
1
>>> boo2 = AnotherValue()
>>> boo2.get_ones()
1
# Deleting a base class with a method does not prevent the creation of new derived classes,
and the method is still saved even after deleting the original base class

>>> class Value:
...     def __init__(self, arg1):
...             self.arg1 = arg1
...     def get_arg(self):
...             return self.arg1
...
>>> class AnotherValue(Value):
...     def __init__(self, arg1, arg2):
...             self.arg1 = arg1
...             self.arg2 = arg2
...     def get_arg2(self):
...             return self.arg2
...
>>> boo = AnotherValue(1, 2)
>>> boo.get_arg2()
2
>>> boo.get_arg()
1
>>> del Value
```

```
>>> boo2 = AnotherValue(2,3)
>>> boo2.get_arg()
2
>>> boo2.get_arg2()
3
# Once again shows that deleting the base class does not prevent the use of methods found in the base class
```

When deleting a class definition where no base class is specified, the delete statement just deletes the class definition itself. Therefore, you are not allowed to make more objects of that same class, while existing objects of that class still exist and perform any actions that are required of it.
When deleting a base class with a derived class, the del operator does not prevent the creation of new derived classes, but prevents the creation of new base classes. It also does not delete the methods found in the base class from the derived class, so methods and attributes found in the base class can still be used and manipulated after the creation of a derived class.