Answers for Debugging Exercises: Chapter 10

Find the Output

1.
```
print(isinstance("Python",object))
```
Ans. True

2.
```
class Parent:
    def func(self):
        print("PARENT func()")
class Child(Parent):
    pass
P = Parent()
C = Child()
P.func()
C.func()
```
Ans.
```
PARENT func()
PARENT func()
```
3.
```
class Parent:
    def func(self):
        print("PARENT func()")
class Child(Parent):
    def func(self):
        print("CHILD func()")
P = Parent()
C = Child()
P.func()
C.func()
```
Ans.
```
PARENT func()
CHILD func()
```
4.
```
class Parent(object):
    def func(self):
        print("PARENT func()")
class Child(Parent):
    def func(self):
        print("CHILD, BEFORE PARENT func()")
        super(Child, self).func()
        print("CHILD, AFTER PARENT func()")
P = Parent()
C = Child()
P.func()
C.func()
```

Ans.

```
PARENT func()
CHILD, BEFORE PARENT func()
PARENT func()
CHILD, AFTER PARENT func()
```

5.

```
class Parent:
    def func1(self):
        print("PARENT func1()")

    def func2(self):
        print("PARENT func1()")

    def func3(self):
        print("PARENT func3()")

class Child(Parent):

    def func1(self):
        print("CHILD func1()")

    def altered(self):
        print("CHILD, BEFORE PARENT func3()")
        super(Child, self).func3()
        print("CHILD, AFTER PARENT func3()")
```

```
P = Parent()
C = Child()
P.func2()
C.func2()
P.func1()
C.func1()
P.func3()
C.func3()
```

Ans.

```
PARENT func1()
PARENT func1()
PARENT func1()
CHILD func1()
PARENT func3()
PARENT func3()
```

6.

```
class Base(object):
    def func1(self):
        print("BASE func1()")
    def func2(self):
        print("BASE func2()")
    def func3(self):
        print("BASE func3()")
class Derived(object):
    def __init__(self):
        self.base = Base()
    def func2(self):
        self.base.func2()
    def func1(self):
        print("CHILD func1()")
    def func3(self):
        print("CHILD, BEFORE OTHER altered()")
        self.base.func3()
        print("CHILD, AFTER OTHER func3()")
C = Derived()
C.func2()
```

```
C.func1()
C.func3()
```

Ans.

```
BASE func2()
CHILD func1()
CHILD, BEFORE OTHER altered()
BASE func3()
CHILD, AFTER OTHER func3()
```

7.

```
class Base:
   bVar = 10
   def __init__(self):
      print("Calling parent constructor")
   def func1(self):
      print('Calling parent method')
   def setVar(self, var):
      Base.bVar = var
   def getVar(self):
      print("Base Variable :", Base.bVar)
class Derived(Base):
   def __init__(self):
      print("Calling Derived Constructor")
   def func2(self):
      print('Calling Derived method')
D = Derived()
D.func2()
D.func1()
D.setVar(20)
D.getVar()
```

Ans.

```
Calling Derived Constructor
Calling Derived method
Calling parent method
Base Variable : 20
```

8.

```
class Base:
    def func(self):
        print('Calling base method')
class Derived(Base):
    def func(self):
        print('Calling Derived method')
D = Derived()
D.func()
```

Ans. Calling Derived method

9.

```
class One(object):
    def __init__(self):
        print("init of One")

class Two(object):
    def __init__(self):
        print("init of Two")

class Three(One):
    def __init__(self):
        print("init of Three")
        super(Three, self).__init__()
class Four(Three, Two):
    def __init__(self):
        print("init of Four")
        super(Four, self).__init__()
F = Four()
```

Ans.

```
init of Four
init of Three
init of One
```

10.

```
class Vehicle:
    def __init__(self, name, color):
        self.__name = name
        self.__color = color
     def get(self):
        return (self.__name, self.__color)
     def set(self, name, color):
        self.__name = name
        self.__color = color
 class Car(Vehicle):
    def __init__(self, name, color, model):
        Vehicle.__init__(self,name, color)
        self.__model = model
     def getDescription(self):
        return self.get(),  self.__model
 C = Car("Ecosport", "Red", "2016")
print(C.getDescription())
```

Ans.

```
  (('Ecosport', 'Red'), '2016')
```

11.

```
class BaseClass1():
    def method_base1(self):
        print("Base 1 method called")
class BaseClass2():
    def method_base2(self):
        print("Base 2 method called")
class DerivedClass(BaseClass1, BaseClass2):
    def derived_method(self):
        print("child method")
D = DerivedClass()
D.method_base1()
D.method_base2()
```

Ans.

```
Base 1 method called
Base 2 method called
```

12.

```
class Parent():
    def __init__(self):
        self.__x = 1
    def show(self):
        print("Show from Parent : ", self.__X)
```

```
class Child(Parent):
    def __init__(self):
        self.__y = 1
    def show(self):
        print("Show from Child", self.__y)
C = Child()
C.show()
```

Ans.

```
Show from Child 1
```

13.

```
class A:
    def method1(self):
        print('Hello...')
class B(A):
    def method2(self):
        print('\t World...')
class C(B):
    def method3(self):
        print('\t\t Good Morning...')
c = C()
c.method1()
c.method2()
c.method3()
```

Ans.

```
Hello...
    World...
          Good Morning...
```

14.

```
class A:
    def display(self):
        print('Hello...')
class B(A):
    def display(self):
        print('\t World...')
class C(B):
    def display(self):
        print('Good Morning...')
c = C()
c.display()
```

Ans.

```
Good Morning...
```

15.

```
class Country:
    def __init__(self, name):
        self.name = name
    def capital(self):
        raise NotImplementedError("Subclass must implement abstract
method")
class India(Country):
    def capital(self):
        return 'New Delhi'
class USA(Country):
    def capital(self):
        return 'Washington DC'
countries = [India('India'), USA('USA')]
for country in countries:
    print(country.name + ': ' + country.capital())
```

Ans. India: New Delhi   USA: Washington DC

16.

```
class One:
    def method1(self):
        print("ONE")
class Two(One):
    def method2(self):
        print("TWO")
class Three(Two):
    def method3(self):
        print("THREE")
T=Three()
T.method1()
T.method2()
T.method3()
```

Ans.

```
ONE
TWO
THREE
```

17.

```
class One:
    def method(self):
        print("ONE")
class Two(One):
    def method(self):
        print("TWO")
class Three(Two):
    def method3(self):
        print("THREE")
T=Three()
T.method()
```

Ans. TWO

Find the Error

1.

```
class One:
    def __init__(self):
        print("init of One")
        super(One, self).__init__()
class Two:
    def __init__(self):
        print("init of Two")
        super(Two, self).__init__()
class Three(One):
    def __init__(self):
        print("init of Three")
        super(Three, self).__init__()
class Four(Three, Two):
    def __init__(self):
        print("init of Four")
        super(Four, self).__init__()
if __name__ == '__main__':
    Four()
```

Ans. `TypeError: super() argument 1 must be type, not classobj`

2.

```
class One(object):
    def save(self):
        super(One, self).save()
class Two(object):
    def save(self):
        super(Two, self).save()
class Three(One):
    def save(self):
        super(Three, self).save()
class Four(Three, Two):
    pass
if __name__ == '__main__':
    Four().save()
```

Ans. `AttributeError: 'super' object has no attribute 'save'`

3.

```
class One:
    def method1(self):
        print("ONE")
class Two(One):
    def method2(self):
        print("TWO")
class Three(Two):
    def method3(self):
        print("THREE")
T=Three()
T.method()
```

Ans. `AttributeError: Three instance has no attribute 'method'`

4.

```
class One:
    def method1():
        print("ONE")
class Two(One):
    def method2():
        print("TWO")
T=Two()
T.method2()
```

Ans. `TypeError: method2() takes no arguments (1 given)`

5.

```
class One:
    def __method(self):
        print("ONE")
class Two(One):
    def __method(self):
        print("TWO")
T=Two()
T.method()
```

Ans. `AttributeError: Two`