Assignment 23 Solutions

1. What is the result of the code, and why?

```
>>> def func(a, b=6, c=8):
print(a, b, c)
>>> func(1, 2)
```

ANS: Here as we have hardcoded and sent 2 values therefore preference is given to the input user gives. so a=1,b=2,c as by default argument 8.

In [1]:

```
def func(a,b=6,c=8):
    print(a,b,c)
func(1,2)
```

```
1 2 8
```

2. What is the result of this code, and why?

```
>>> def func(a, b, c=5):
print(a, b, c)
>>> func(1, c=3, b=2)
```

ANS: The result of the above code is `1 2 3` . It is because the function will use default values only when a value for a argument is not provided and if the argument name is mentioned while doing a function call, the order of arguments is also ignored by the python interpreter

In [3]:

```
def func(a,b,c=5):
    print(a,b,c)
func(1,c=3,b=2)
```

```
1 2 3
```

3. How about this code: what is its result, and why?

```
>>> def func(a, *pargs):
print(a, pargs)
>>> func(1, 2, 3)
```

ANS: The result of the code is `1 (2,3)` . `*pargs` stands for variable length arguments. This format is used when we are not sure about the number of arguments to be passed to a function. All the values under this argument will be stored in a tuple.

In [4]:

```
def func(a, *pargs):
    print(a,pargs)
func(1,2,3)
```

```
1 (2, 3)
```

4. What does this code print, and why?

```
>>> def func(a, **kargs):
print(a, kargs)
>>> func(a=1, c=3, b=2)
```

ANS: The result of the above code is `1 {'c': 3, 'b': 2}` . `**args` stands for variable length keyword arguments. This format is used when we want pass key value pairs as input to a function. All these key value pairs will be stored in a dictionary

In [5]:

```
def func(a,**kargs):
    print(a,kargs)
func(a=1,c=3,b=2)
```

```
1 {'c': 3, 'b': 2}
```

5. What gets printed by this, and explain?

```
>>> def func(a, b, c=8, d=5): print(a, b, c, d)
>>> func(1, *(5, 6))
```

ANS: `*` is the unpacking operator and are operators that unpack the values from iterable objects in Python. The single asterisk operator `*` can be used on any iterable that Python provides, while the double asterisk operator `**` can only be used on dictionaries. In the example the value `*(5,6)` will be unpacked and will be assigned to b and c and passed as arguments, `d =5` will taken by defaults are keyword arguments.

The output of the above is 1 5 6 5.

In [6]:

```
def func(a, b, c=8, d=5):
    print(a, b, c, d)
func(1, *(5, 6))
```

```
1 5 6 5
```

6. what is the result of this, and explain?

```
>>> def func(a, b, c): a = 2; b[0] = 'x'; c['a'] = 'y'
>>> l=1; m=[1]; n={'a':0}
>>> func(l, m, n)
>>> l, m, n
```

ANS: The output of above code is `1, ['x'], {'a': 'y'}` .

1. Even though Python gives importance to indentation, it provides a facility to declare an entire function in one single line where statements in a function body are sepereated by `;`
2. When `l,m,n` are provided as inputs to the function, it modifies the values of l,m,n and sets the value of `l=2` , `m=['x']` and `n={'a':'y'}`

In [7]:

```
def func(a, b, c): a = 2; b[0] = 'x'; c['a'] = 'y'
l=1; m=[1]; n={'a':0}
func(l, m, n)
l,m,n
```

Out[7]:

```
(1, ['x'], {'a': 'y'})
```