

## Solutions, Week 2, Day 3

1. What names are defined in each of three files A, B and C given that:

- A defines `searchdb`, `_calcCache`
- B defines `_calcCache`, `find`
- C defines `taylorSeries`, `__initTable__`
- A has `import B`
- B has `from C import *`
- C has `import urlparse`

It's easiest to work backwards through the chain of import statements. And remember that underscored names aren't imported in an `import *` statement. C has the names `taylorSeries`, `__initTable__` and `urlparse`. B has `_calcCache`, `find`, `taylorSeries` and `urlparse`. A has `searchdb`, `_calcCache` and B.

2. What if the import statements were instead:

- A has `import B, C`
- B has `import C` and `from C import taylorSeries`

C is as above. B has `_calcCache`, `find`, C and `taylorSeries`. A has `searchdb`, `_calcCache`, B and C.

3. What does the list comprehension `[(x,y) for x in range(3) for y in range(3)]` produce? Why?

`[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)]`. It is effectively saying:

```
theLst = []
for x in [0, 1, 2]:
    for y in [0, 1, 2]:
        theLst.append((x,y))
```

4. `[(x,y) for x in range(3) for y in range(3) if (x < y)]`?

`[(0, 1), (0, 2), (1, 2)]` It is just like the above list except only certain elements are placed in the list:

```
theLst = []
for x in [0, 1, 2]:
    for y in [0, 1, 2]:
        if (x < y):
            theLst.append((x,y))
```

5. `[(x,y) for x in range(3) if (x < y) for y in range(3)]`?

Depends. The function is effectively:

```

theLst = []
for x in [0, 1, 2]:
    if (x < y):
        for y in [0, 1, 2]:
            theLst.append((x,y))

```

It could generate an error because the first time through the loop ( $x < y$ ) is tested before  $y$  has been defined. However if  $y$  has been used before in the same python session, it will have a value and the results are unpredictable, ranging from the full list as above to an empty list.

- Write a recursive function that sorts a list. Hint: find the lowest number, then the second lowest and so on.

This isn't an efficient sort, but illustrates the principle. We find the first (lowest) number and then construct a list starting with it and followed by the remainder of the list being sorted:

```

def recursiveSort (lst):
    if (len(lst) == 1):
        return lst
    else:
        theMin = min (lst)
        lst.remove (theMin)
        return [theMin] + recursiveSort (lst)

```

- Write a function that accepts a string (`times`, `divide`, `plus` or `minus`) and returns a function that calculates that operation over two arguments. For example, `times` should return a function of the form  $f(a,b)$  that calculates  $a*b$ .

Assuming for the purposes of this exercise, that the input is valid:

```

def genFunc (op):
    if (op == "times"):
        return lambda a, b: a*b
    elif (op == "divide"):
        return lambda a, b: a/b
    elif (op == "plus"):
        return lambda a, b: a+b
    elif (op == "minus"):
        return lambda a, b: a-b

```

- Write a function that accepts strings of the form  $x \text{ op } y$  where  $x$  and  $y$  are integers and `op` is one of `times`, `divide`, `plus`, `minus`. The function should return the result of the calculation. For example, `12 times 13` will return 156.

```

def calcOp (opstr):
    thelst = opstr.split()
    val1, op, val2 = thelst[0], thelst[1], thelst[2]
    if (op == "times"):
        return val1*val2
    elif (op == "divides"):
        [...]

```