nested control structures :

To solve a problem we may have to use for within for, for within while, while within while, while within for and so on. We will look at examples requiring such constructs.

1) Generate numbers the way they are on a snake-ladder board.

100 99 98 ... 91
 81 82 83 ... 90
...
...
 20 19 ...   11
  1 2 ...   10

Can we find some pattern in the display.
In the first row, the numbers are in decreasing order.
In the second row, the numbers are in the increasing order.
Let us generate the first two rows.
We shall use a for loop as the # of iterations is known.

```
# file : 1a_snake_ladder.py
# version 1
# Generate the first two rows only
for i in range(100, 90, -1):
        print(i, end = " ")
print()
for i in range(81, 91) :
        print(i, end = " ")
print()
```

We shall make two changes to this version.
In stead of hard coding the values of the range like 100 and 81,
we shall use some variables.
we shall make each number get displayed using a width of 3.

```
# file : 1b_snake_ladder.py
# version 2
# Generate the first two rows only
# use variables instead of constants
# adjust the spacing
n = 100
for i in range(n, n - 10, -1):
        print("{0:3}".format(i), end = " ")
print()
n -= 20
for i in range(n, n + 10) :
        print("{0:3}".format(i + 1), end = " ")
print()
```

We shall generate all the rows by repeating this code 5 times.

```
# file : 1c_snake_ladder.py
# version 3
# Generate all rows; repeat the earlier code 5 times.
n = 100
for j in range(5) :
        for i in range(n, n - 10, -1):
                print("{0:3}".format(i), end = " ")
        print()
        n -= 20
        for i in range(n, n + 10) :
                print("{0:3}".format(i + 1), end = " ")
        print()
```

This is the final output.

```
100  99  98  97  96  95  94  93  92  91
 81  82  83  84  85  86  87  88  89  90
 80  79  78  77  76  75  74  73  72  71
 61  62  63  64  65  66  67  68  69  70
 60  59  58  57  56  55  54  53  52  51
```

```
 41  42  43  44  45  46  47  48  49  50
 40  39  38  37  36  35  34  33  32  31
 21  22  23  24  25  26  27  28  29  30
 20  19  18  17  16  15  14  13  12  11
  1   2   3   4   5   6   7   8   9  10
```

2) Display the following patterns.

for n = 4,

```
             1   =  1
         1 + 2  =  3
     1 + 2 + 3   =  6
 1 + 2 + 3 + 4 = 10
```

Let us analyze the problem.
The last row indicates that we have to display numbers from 1 to n and display the sum with + or = placed between the numbers.
We have to repeat that for all values of 1 to n.
Let us try to generate the last row without the symbols + and =.
We will use a for loop as the number of iterations is known.
As in any summation problem, we shall initialize the sum to 0 before the loop and increment each time within the body of the loop.

```
# file 2a_sum.py
# Generate the following pattern for n = 4
# 1 2 3 4 10
n = int(input("enter an integer : "))
s = 0
for i in range(1, n + 1):
        s += i
        print("{0:3}".format(i), end = " ")
print("{0:4}".format(s))
```

This is the output at this stage.

```
$ python 2a_sum.py
```

```
enter an integer : 4
  1   2   3   4   10
```

We will try to get the symbol + or =. We observe that the number of + should be one less than n. So we cannot blindly display + in the for loop. One of the possible solutions is to compare i with n to decide whether to display + or =. This is a bad idea as we will introduce unnecessary comparison within the loop.
Another solution would be to execute the loop n - 1 times and take care of the last addition outside the loop.
Another solution would be display 1 before we enter the loop and then execute the loop n - 1 times.
We shall try the last possibility.

```
# file 2b_sum.py
# Generate the following pattern for n = 4
# 1 + 2 + 3 + 4 = 10
n = int(input("enter an integer : "))
s = 1
# take care of 1 before the loop
print("{0:3}".format(1), end = " ")
for i in range(2, n + 1): # do n - 1 times
        s += i
        print(" +", "{0:3}".format(i), end = " ")
print(" =", "{0:3}".format(s))
```

This is the output at this stage.

```
$ python 2b_sum.py
enter an integer : 4
  1  +   2  +   3  +   4  =   10
```

We will try to generate n such rows. We will put all this code within a for loop. At this point all the output shall be left aligned.

```
# file : 2c_sum.py
```

```
# wrong program !!
# Generate the following pattern for n = 4
# 1 = 1
# 1 + 2 = 3
# 1 + 2 + 3 = 6
# 1 + 2 + 3 + 4 = 10
m = int(input("enter an integer : "))
s = 1
for n in range(1, m + 1):
        print("{0:3}".format(1), end = " ")
        for i in range(2, n + 1): # do n - 1 times
                s += i
                print(" +", "{0:3}".format(i), end = " ")
        print(" =", "{0:3}".format(s))
```

```
$ python 2c_sum.py
enter an integer : 4
  1  =   1
  1  +   2  =   3
  1  +   2  +   3  =   8
  1  +   2  +   3  +   4  =  17
```

But the output is apparently wrong. Take a few minutes to think before you proceed further.

If we already have the sum of first i numbers, we should add i + 1 to get the sum of the first i + 1 numbers. The summation should be moved to the outer loop.

This is the corrected program.

```
# file 2d_sum.py
# Generate the following pattern for n = 4
# 1 = 1
# 1 + 2 = 3
```

```
# 1 + 2 + 3 = 6
# 1 + 2 + 3 + 4 = 10
m = int(input("enter an integer : "))
s = 0  # observe here
for n in range(1, m + 1):
        print("{0:3}".format(1), end = " ")
        for i in range(2, n + 1):
                print(" +", "{0:3}".format(i), end = " ")
        s += n # observe here
        print(" =", "{0:3}".format(s))
```

This is the output.

```
$python 2d_sum.py
enter an integer : 4
  1  =   1
  1  +   2  =   3
  1  +   2  +   3  =   6
  1  +   2  +   3  +   4  =  10
```

As the last step, we will increase spacing before the first column. Decrease the spacing for the subsequent rows.

```
# file 2e_sum.py
# Generate the following pattern for n = 4
#                   1 = 1
#             1 + 2 = 3
#       1 + 2 + 3 = 6
# 1 + 2 + 3 + 4 = 10
m = int(input("enter an integer : "))
s = 0
for n in range(1, m + 1):
        # generate spaces proportional to m - n
        # spacing should decrease as n increases
```

```
        print("       " * (m - n), end = " ")  # why 7 spaces ?

        print("{0:3}".format(1), end = " ")
        for i in range(2, n + 1):
                print(" +", "{0:3}".format(i), end = " ")
        s += n
        print(" =", "{0:3}".format(s))
```

This is the output.

```
$ python 2e_sum.py
enter an integer : 4
                     1  =  1
               1  +  2  =  3
         1  +  2  +  3  =  6
   1  +  2  +  3  +  4  = 10
```

---

3) Write a program to find the highest number of factors in the range of numbers from 1 to n.

Let us write a program to find the number of factors of a given number. Then we will try to find the number of factors of all numbers from 1 to n. Then we will find the highest amongst them.

To find the number of factors of a given number, we can use a naive for loop which executes n times or a while loop which executes sqrt(n) times. We will prefer the latter.

```
# file 3a_count_factors.py
# count the number of factors
n = int(input("enter an integer : "))
c = 0 # count of factors
i = 1
```

```
while i * i < n :
        if n % i == 0 :
                c += 2 # think: why 2 and not 1
        i += 1
# to take care of perfect square
if i * i == n :
        c += 1
print(c)
```

The outputs:

```
$ python 3a_count_factors.py
enter an integer : 28
6
$ python 3a_count_factors.py
enter an integer : 25
3
```

Let us generate for all numbers from 1 to n.
Introduce an outer for loop to execute n times.

```
# file 3b_count_factors.py
# count the number of factors of numbers from 1 to n
n = int(input("enter an integer : "))
for m in range(1, n + 1):
        c = 0
        i = 1
        while i * i < m :
                if m % i == 0 :
                        c += 2
                i += 1

        if i * i == m :
                c += 1
        print(m, "=>", c)
```

The output is:

```
$ python 3b_count_factors.py
enter an integer : 10
1 => 1
2 => 2
3 => 2
4 => 3
5 => 2
6 => 4
7 => 2
8 => 4
9 => 3
10 => 4
```

Now let us track the highest so far. We will assume it to be zero initially. we will update it if any number between 1 to n has number of factors higher than the highest so far.

```
# file 3c_count_factors.py
# count the number of factors of numbers from 1 to n; find the highest
n = int(input("enter an integer : "))
max_count = 0
for m in range(1, n + 1):
        c = 0
        i = 1
        while i * i < m :
                if m % i == 0 :
                        c += 2
                i += 1

        if i * i == m :
                c += 1
        if c > max_count:
```

```
                max_count = c
print("max # of factors : ", max_count)
```

The output :

```
$ python 3c_count_factors.py
enter an integer : 100
max # of factors :  12
```

I do not know whether it is right. You may want to test it thoroughly.