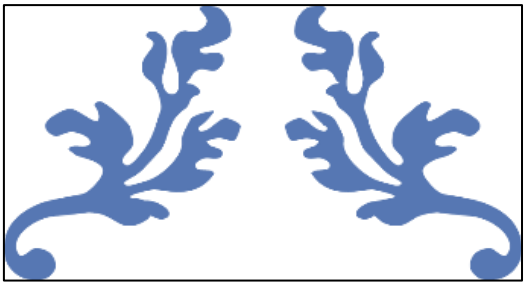

SCHUR'S NUMBER S(n)

Project n°1 : PYTHON

Lise SPILERS – Monica-Pauline BLANC

INT2

Efrei PARIS

3.2. Warm up exercise :

1) Write a program that colours, and displays, the N first integer numbers with two colours s.t., the even numbers are coloured in red, and the odd numbers are coloured in blue.

First, we secure the program in order to repeat the question ("How many integer do you want") for all N≤ 0 because we can't have a negative value or equals to 0 (it will directly equals to 0). We already know that an even number is a number that can be divided by 2. So we use the *modulo* by 2 in a *for loop* (We use the modulo in order to have just the rest of the Euclidian division). In the *for loop* we insert an *if* in the case that the *modulo* is equals to 0 and so the number is even (so we color the number in blue) and an *else* in the other case, so the number is odd (we color it in red).

First, we secure the program in order to repeat the question ("How many integer do you want") for all N≤ 0 because we can't have a negative value or equals to 0 (it will directly equals to 0). We already know that an even number is a number that can be divided by 2. So we use a *for loop* in order to color all number (from one to the requested number by the user).

Then, still in the *for loop*, we use an *if* condition with the *modulo* by 2 equal to 0 (We use the modulo in order to have just the rest of the Euclidian division in this case), it mean that the number it's even so then we color it in red. In the other case (*else* condition), the number is odd so we color it in blue.

```
Algorithm: 3.2.1
VAR : N,i : integer
      Colour [B,R] : array of colors
      Red, Blue : colors
Begin
      N ← 0
      do
          Write("How many integer do you want to color ?")
          Read(N)
      While (N≤ 0)
      For i ← 1 to (N+1) do
          If (i%2=0) then
              Write(B,i)
          Else
              Write (R,i)
          End if
      End for
end
```

2) Write a program that randomly colours the N first integer numbers using n (the n colours must be present).

First, we secure our program in order to not take negative value for the number of integer and a negative value or above to 6 for the number of colors (above to 6 because we have just 6 colors).

In this exercise, the main difficulty is to be sure that the n requested color are present.
To begin we create a list with all colors : white, red, green, orange, blue and purple.

In a second time, we aske the user to enter how many colors he wants, and then we create an other *list*, we named it "col", with just the number of requested colors by the user. The program will choose the n first color in the list "color" to compose the new list "col", for this we use : "col = color [:n]"

Now, we want to make sure that all requested colors are present. To do so,we use a *for loop* to color all number one by one and we create a new variable (we choosed "a") which count how many elements are in the list "col" thanks to a method of list objects : *len(col)*

Then we use an *if* condition in case of the variable "a" is equal to, so it mean in case of the list "col" is empty. In this case, we fill again the list "col" with just the number of requested colors (we use again "col=color[:n]"), the program choose a color in the list "col", then color the number before to print it, and the color used is deleted. For the other case (the list is not empty), we have an *else* condition. In this case, the program color the number, print it and deletes the color used of the list "col". Thanks to this method, we can be sure that all color will be used because all color already used are deleted until there are no more in the list.

```
Algorithm: 3.2.2
VAR :  N,n,i: integer
       White, Red, Green, Orange,Blue, Purple : colors
       Colour = [W,R,G,O,B,P] : array of colors
       Col = array of colors
       Empty[ ] = array
Begin
    |  n ← 0
    |  do
    |        Write("How many colors do you want below or equals to 6 ?")
    |        Read(n)
    |  While (n≤ 0) OR (n> 6)
    |  Col ← Colour [ : n]
    |
    |  N ← 0
    |  Do
    |  |     Write ("How many integer do you want to be colored (above or equals to n)?")
    |  |     Read(N)
    |  While (N< n)
    |
    |  For i ← 1 to n+1 do
    |        if (Col=Empty) then
    |  |           Col ← Colour [:n]
    |  |           A.random.choice(Col)
    |  |           Write (A,i)
    |  |           Col ← Col - A
    |        Else
    |  |           A ← random.choice(Col)
    |  |           Write (A,i)
    |  |           Col ← Col - A
    |        End if
    |  End for
End
```

3.3. Some basic function :

1) Write a program check if a given coloured triplet is monochromatic or not. The input of this program is a triplet and its colouring.

We secure the program so as to not have $x \leq 0$ or $y \leq 0$ or $z \leq 0$ because x,y,z are the value of the triplet : we can't have negative number. Then we ask to the user the color of each number in the triplet. We decide to create a list called "color" to have for each color the first letter. So, we ask to the user to directly write the first letter of the color (for white, the user writes W) and for each letter it fill an other variable (A,B,C : they are string). Then we see if the color (W or B or P …) is in the list "color". If it is, we can use a methods of list object : list.insert(i, x). It serves to insert an item at a given position. The first argument (i) is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list. However, in this case, we insert associate two list "initcolor" (it contains the code of the color) and "color" (it contains just the first letter. So, when the user write W the program associate W to '\33[0m'. Then, we can see that if A=B and B=C then the triplet is automatically monochromatic (the three color are the same) and we print the triplet with corresponding color. Otherwise, it's not monochromatic and we display the triplet with the corresponding color.

We need to secure the program thanks to two *while* conditions:

- In case that the user enter wrong values, so as to not have $x \leq 0$ or $y \leq 0$ or $z \leq 0$ because x,y,z are the value of the triplet : we can't have negative number and $x \leq y$ and $z=x+y$ (according to the definition of a triplet.
- In case that the user enter a wrong letter or forget that it must be a capital letter (concerning the colors of each number of the triplet).

First we created all colors and put all of them in a list called "initcolor". Like we need to aks the user to enter the colors of his triplet, we decide to create a list called "color" that contain the initial letter of each color. So, we ask to the user to directly write the first letter of the color (for white, the user writes W) and for each letter it fill an other variable (A,B,C : they are string).

Then we see if the color (W or B or P …) is in the list "color". If it is, we can use a methods of list object : list.insert(i, x). It serves to insert an item at a given position. The first argument (i) is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list. However, in this case, we insert associate two list "initcolor" (it contains the code of the color) and "color" (it contains just the first letter. So, when the user write W the program associate W to the white code : '\33[0m'.

To finish, we check if A=B and B=C. In this case, the triplet is automatically monochromatic (the three color are the same) and we print the triplet with corresponding color. Otherwise, it's not monochromatic and we display the triplet with the corresponding colors.

Algorithm: 3.3.1
VAR : x,y,z : integer
A,B,C : str
White, Red, Green, Orange,Blue, Purple : colors
initcolour = [W,R,G,O,B,P] : array of colors
Color :["White", "Red"", "Green," "Orange", "Blue", "Purple"] : array of colors
Begin
Write("Enter your triplet :")
Read(x,y,z)
Write("What is the color of each number of your triplet ? ")
Write ("You have the choice with white, blue, red, green, orange, purple"
Write ("Write the color of your choice")

2) Write a program that generates all triplets (a,b,a+b) avec a, b, a+b for a, b, a+b ≤ p, for a given value p.

Like in our precedent program, we start by securing the program with a *while* condition because we can not have a negative value. When the user has entered its value, the program will generate all triplet : we know a must be inferior than b or equal to be, and the maximum value of a+b is p (the value entered by the user). Thereby we begin with a and b equal to one (because it the first the can have) and we a *while loop* to …

In the while loop we use an *if* condition : we begin with a=1, so b will be take all possible values from 1 until a + b = p, and we print all triplet one by one. Then thanks to *else*, when a+b=p, we can add 1 to the value of a and like b must be equal or superior than a, we begin our new *loop* with the same value for a and b (b take the new value of a).

```
Algorithm: 3.3.2
VAR :  p,a,b : integer
Begin
        a ← 1
        b ← 1
        p← 0
        Do
            Write("For a given value p, the program will generate all triplets. Enter p:")
            Read(p)
        While (p≤ 0)
        Write("The list of all triplets is :")
        While (a≤p) and (b≤p)
            If (a+b≤p) then
                Write ("(",a,b,a+b,")")
                b ← b +1
            end if
            if (b≥p) then
                a ← a +1
                b ← a
            end if
        end while
end
```

3) Write a program that randomly colours N first integer numbers, for a given N. Then, the program evaluates the predicate and displays the result i.e., it displays TRUE if all triplets are not monochromatic, and FALSE otherwise.

At the beginning, we compared the colors (A, B, and C) of the triplets after the *loop while*, but we have understood that using this method only allowed us to compare the last A, B and C, so just the colors in the last triplet.
So we decide to use an other method : before the *while loop*, we created a list named "monochromatic". Then, after each new triplets A, B and C, we compare it, and use an *if* condition : if all of them have the same color, it means that we will have a monochromatic triplet, so we fill the list "monochromatic" with one of the letter (we choose one).
The goal is just to empty the list, because at the end of the program, we look at the list :
We use the method of list objects : *len* to count how many elements are contained in the list and then we use an *if* condition : if the list is empty, it mean that no one triplet is monochromatic, so we can print " True", in the other case, we print "false".

```
Algorithm : 3.3.3
VAR :  Red,Green,White : colors
       Color = [red,green] = array of color
       a,b,i ,N : integer
       test : Boolean
       monochromatic [ ] : array of integer
       A,B,C : colours
       Empty[ ] : empty array

Begin
    N← 0
    a← 0
    b← 0
    test ← TRUE
    Do
        Write ("Enter the number of integer you want")
        Read (N)
    While (N≤ 0)
    Write ("The list of all the triplets is :")
    For i ← 1 to (N+1) do
        While ( a+b≤ N )
            A= random.choice (color)
            B = random.choice (color)
            C = random.choice(color)
            If (A=B) AND (B=C) then
                Monocromatic [ ] ← A
            End if
            If ( a+b≤ N ) then
                Write ("(",A,a,B,b,C,a+b,")")
                b ← b+1
            end if
            if (b ≥ N)
                a ← a+1
                b← a
            end if
        end while

    end for
```

```
        if (monocromatic = empty) then
            write ("None are monochromatic")
        else
            write ("All or one triplet(s) are/is monochromatic(s)")
            test ← FALSE
        end if
        write (W,"test")
end
```

4) Write a program that colours the N first numbers with n colours, and checks whether the n colours are present in the colouring.

First we creat the color thanks to "random" and put all of them in a list called "color". Then we secured our program with a while loop because number of integers must be superior or equal to the number of colors.

Then we use a *for loop* to color all integer, from one to N, one by one, and when the number is colored, we put it in a list called "choiced2"

To finish, thanks to the six variable (A, B, C, D, E and F) and the method of list object *choice2.count*, we count how many times each color appears in the list choice2. We put them in a new list "G", so we have as much element in the list G as colors, and we remove all 0 which are in the list G, (so we remove an element for each color which is not in the list choice2) . So now, we just have to count how many elements contains the list G thanks to the method of list object *len(G)* and if there is as much element in the list as requested colors by the user, it's valid, in the other case, it is not valid : so we use an *if* condition and an *else*.

```
Algorithm : 3.3.4
VAR :  Red,Green,White,Orange,Blue,Purple : colors
       Choice : colors
       Color = [red,green,white,orange,blue,purple] = array of color
       a,b,i ,N, n, A,B,C,D,E,F : integer
       G [ ] : array of integer
       Col [ ] : array of colours
Begin
       N← 0
       a← 1
       b← 1
       n ← 0
       Do
           Write ("How many integer do you want ?")
           Read (N)
           Write ("How many int
       While (N≤ 0)
       Write ("The list of all the triplets is :")
       For i ← 1 to (N+1) do
           Choice ← random.choice(col)
           Write (choice, i)
           A= random.choice (color)
           B = random.choice (color)
           C = random.choice(color)

       End for
       While (0 in G)
           If 0 in G then

       End while
```

```
        If
            | Write (W,"The colouring is valid")
        Else
            | Write (W,"The colouring is not valid")
        End if
end
```

5) TQ: Who many different colourings are possible to colour N numbers with 2 colours?

If we have 2 colors, we know that we have to put in power the number of integer: so in order to know how many different colourings are possible to colour N number with 2 colors, we have to do : 2^N

6) TQ: Same question for n colours ?

If we have n colors, we have to do : n^N

7) Write a program that converts a decimal number to a binary one (e.g., 11à1011).

First we ask the user to enter its number in base 10.
In a second time, we know that 0 is equal to 0 in all base, so we use a "if" condition for this particular case, and an "else" condition for all other cases. Then, for all the other cases, we know that to converse a decimal number to binary, we have to do successives division: we divide the number by 2, the remainder is 1 or 0, then we devide the quotient of the precedent division by 2, and we obtain a remainder equal to 1 or 0 again… At the end, to have the decimal number in binary, we just have note all the remainder, from last to first.
To do successives division in python, we used a "while" condition. The remainder is putting in list, because it corresponds to the number in binary. However, as the binary number corresponds to all remainder from the last to the first, when the list is finished, we have to reverse it. Now, we can return the binary number to the user.

```
Algorithm : 3.3.7
VAR :  number, quotient, remainder = integer
       Restofthedivision [ ] = array of integer
       Restofthedivision2 [ ] : array of integer
Begin
    |  Number← -1
    |  Do
    |      |  Write ("Enter your decimal number")
    |      |  Read (number)
    |  While (number<0)
    |  If (number=0) then
    |      |  Write ("Your number in binary system is 0")
    |  Else
    |      |  Write ("Your number in binary system is :")
    |      |  While (number ≠ 0)
    |      |      |  Quotient ← number // 2
    |      |      |  Remainder ← number % 2
    |      |      |  Number ← quotient
    |      |      |  Restofthedivision [ ] ← reaminder
    |      |  End while
    |      |  For i← restofthedivision[-1] to restofthedivision [0]
    |      |
    |      |
    |
```

```
                    Restofthedivison2 [ ] ← restofthedivision [i]
                End for
                Write (restofthedivision2)
        End if
End
```

8) Generalise the previous program to convert a decimal number to a number in a base b (>1).

For this exercice, we did two programs because we didn't know for which base the program must work. So the first is for all base until the base 16 because in digital information class we stop at the base 16, and the second works for all base until the base 36 because then, there is no more letter in the alphabet. In the two programs, we use the same way.
First, we ask the user to enter his decimal number and the base he wants. Then, we use the method of successives divisions by the base like in the 3.3.7 (example: 35 in base 8, we do the successive division with dividing "(and then all quotient we find by 8).
However, for all base bigger than 10, the result contains some letter (for example, 30 in base 16 is "1E"). So we use some "if" condition for all remaider bigger than 9 to have a letter.
To finnish, like we did in the 3.3.7, all remaider are putting in a list, then the program reverse the list and return-it to the user.

```
Algorithm : 3.3.8
VAR :  number, quotient, remainder, i = integer
       Restofthedivision [] = array of integer and letter
       r : integer or letter (A,B,C …)
       restofthedivision2 [ ] : array of integer and letter
Begin
       number← -1

       b← 0
       Do
            Write ("Enter your decimal number")
            Read (number)
       While (number<0)
       Do
            Write ("Enter you wanted base >1")
            Read(b)
       While (b≤ 1)
       If (number=0) then
            Write ("Your number in the conversion system  is 0")
       Else
            Write ("Your number in",b,"system is : ")
            While (number≠ 0)
                 Quotient ← number // b
                 Remainder ← number % b
                 Number ← quotient
                 if (r=10):
                      r←"A"
                  if (r=11):
                      r←"B"
                  if (r=12):
                      r←"C"
                  if (r=13):
                      r←"D"
                  if (r=14):
                      r←"E"
```

```
                    if (r==15):
                        r←"F"

                    if (r==16):
                        r="G"
                    if (r==17):
                        r←"H"
                    if (r==18):
                        r←"I"
                    if (r==19):
                        r←"J"
                    if (r==20):
                        r←"K"
                    if (r==21)
                        r←"L"
                    if (r==22):
                        r←"M"
                    if (r==23):
                        r←"N"
                    if (r==24):
                        r←"O"
                    if (r==25):
                        r←"P"
                    if (r==26):
                        r←"Q"
                    if (r==27):
                        r←"R"
                    if (r==28):
                        r←"S"
                    if (r==29):
                        r="T"
                    if (r==30):
                        r="U"
                    if (r==31):
                        r="V"
                    if (r==32):
                        r="W"
                    if (r==33):
                        r="X"
                    if (r==34):
                        r="Y"
                    if (r==35):
                        r="Z"
                    Restofthedivision [ ] ← reaminder
                End while
            End if
            For i← restoftgedivision[-1] to restofthedivision [0] do
                Restofthedivision2 [ ]← restofthedivision [i]
            End for
            Write (restofthedivision)
end
```

9) Write a program to store all the possible colourings of the N first integer

numbers with n colours. The program then displays this list.
Our problem : faire en sorte de pas avoir la même combinaisons

Our big problem was to be sure to have all colored used. So we decide to use the 3.3.8 for a base b. First, we ask to the user to enter a number (this number is going to define the number of bit), then we ask the number of colors (below or equals to six) and it defines the base. We add a variable called num which is corresponds to the number of possible colouring combination. Then we use the 3.3.8 in order to calculate the number in the wanted base (so in the base n).

Then, we do an another for loop to consider all k in a list called 'allnumber". This loop serves to have the same size of all elements in the list "allnumber". It's for that that we insert to the position 0 a 0 so as to complete the list.

For example, if we enter 4 for N and 3 for n (in "allnumber" at the end we have to have 4 numbers) we have in "allnumber" this :

allnumber= [[0], [1], [2], [1, 0], [1, 1], [1, 2], [2, 0], [2, 1], [2, 2], [1, 0, 0], [1, 0, 1], [1, 0, 2]…]

So, for the 9 elements before the first 3 numbers ([1, 0, 0]) we have to add 0 in order to complete and 3 elements.

Finally, we do 2 checks: we check for all l if it is is in "allnumber" and for m if it is in l and if it is we can print. The "for m in l" make a new loop in list of colors.

The print (" ") stops the program and print the numbers in columns

3.4. Application :

1) Based on the previous programs and the problem description (see the table to illustrating S(2)), write a program to compute S(2).

2) TQ: What is the value of S(2) ? Justify your answer.

3) Generalize the previous program to compute S(n) for n > 2.

4) TQ: How many Schur's numbers were you able to calculate? What was the encountered problem ?.

5) Based on the following code, calculate the execution time to compute S(n).
To the 3.4.1 to 3.4.5, we didn't know how to do it. We can see in our file that we do several test but it give nothing. We try to do the best.

4. Advanced functions :

This was simple because we already have the formula ($\frac{3^n-1}{2} \leq S(n) \leq 3 \times n! - 1$) with n≥ 6.
So, we create 2 main variables that will take the value on the the left and on the right. But on the right, we have the factorial of a positive integer n, denoted by n!, and in Python we can't write !n. So, we use a variable called fact to compute !n. We already know that $!n = 1 \times 2 \times 3 \times ... \times n$
So, we use a for loop to say that : for i (initialize to 1) to N+1 (in order to start from 1), compute fact time i . Then we can compute the right part and display both side.

```
Algorithm : 4.1
VAR :  left, right, Fact, n,i : integer
Begin
        n← 0
        Fact← 1
        Do
            |       Write ("Enter n in order to bound S(n) (above or equal to 6)")
            |       Read (n)
        While (n≤6)

        Left ← ((3^n)-1)/2
        For i← 1 to (n+1) do
            |       Fact = fact * i
        End for
        Right ← 3*fact-1
        Write ("The intervals is:", left," ≤S(",n,") ≤", right)
end
```