Lab 3: Python – Loops

Chapter 0: Recap

% Data types (*int*, *float*, *string*, *complex etc*) are implicitly declared by Python
% Python programs can be saved as .py files.

% We can convert *int* or *float* to *str* and vice versa.

% **if**..**else** statement are used to execute a block of statements if condition is true, and another block of statements when the condition is false**.**

Loop Statement

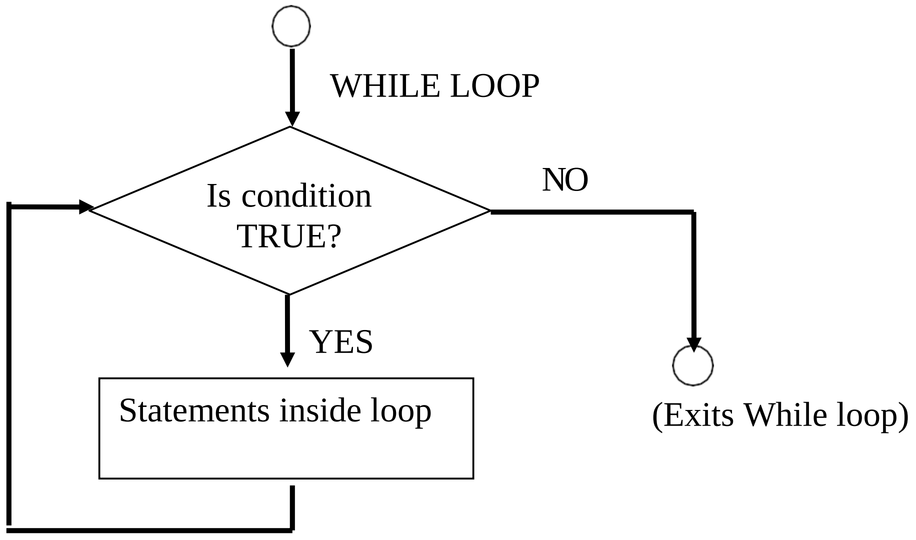
A loop statement allows us to execute a group of statements multiple times.

Python has two types of loops: `While` loop and `For` loop.

Chapter I: While-Loop

1. Basic While-loop

`While`-loop repeatedly executes the target set of statements as long as the given condition is *true*. When the condition becomes *false*, we exit the while-loop



In a blank cell, type the following:

```
n=10
while n > 0:
    print('countdown:', n)
    n=n-1
print('It is done. Now outside the loop.')
```

The colon (**:**) is important.
It marks the beginning of a block of code

Run the code

The above lines of code should give the following output:

```
countdown: 10
countdown: 9
countdown: 8
countdown: 7
countdown: 6
countdown: 5
countdown: 4
```

```
countdown: 3
countdown: 2
countdown: 1
It is done. Now outside the loop.
```

2. Infinite While loop

One of the common errors in programming is caused due to *infinite loop*. Let's try this example. In the earlier code snippet

Delete the line: `n=n-1`

So that the code snippet now looks like this:

```
    n=10
    while n > 0:
        print('countdown: ', n)
    print('It is done. Now outside the loop.')
```

In the code, the value of *n* is always 10. Hence the *while* condition is always true. Hence the loop will run infinitely many times.

Try running the cell

The above lines of code should give the following output:

```
countdown: 10
countdown: 10
countdown: 10
countdown: 10
```

… and so on

CLICK **'Control+M'** to stop the execution of the program; or select menu, Runtime>Interrupt execution

Always check your code to see that in a while loop the condition becomes false eventually.

3. Example: Divisors of a number

Let's write a code snippet to print all the divisors of any input integer.

In a single cell type:

```
print("To find number of divisors other than 1 & itself")

n = input("Enter any positive integer: ")
num = int(n)
i = 2
divisors=0
while i < num:
    if num % i == 0:
        print(i,'is a divisor of', n)
        divisors = divisors+1
    i = i + 1
print('The number of divisors of', n, 'is', divisors)
```

Run the code snippet multiple times; give various values of integers as input and check the output.

<u>Variant</u>: Now, the above code can be improved. For instance, no number greater than *n*/2 will be a divisor of *n*. SO, we don't need to run the loop until *i*=*n*. We can run it only until *i* > *n/2*.

In the *above code* code, replace

```
    while i < num:
```

with

```
    while i <= num/2:
```

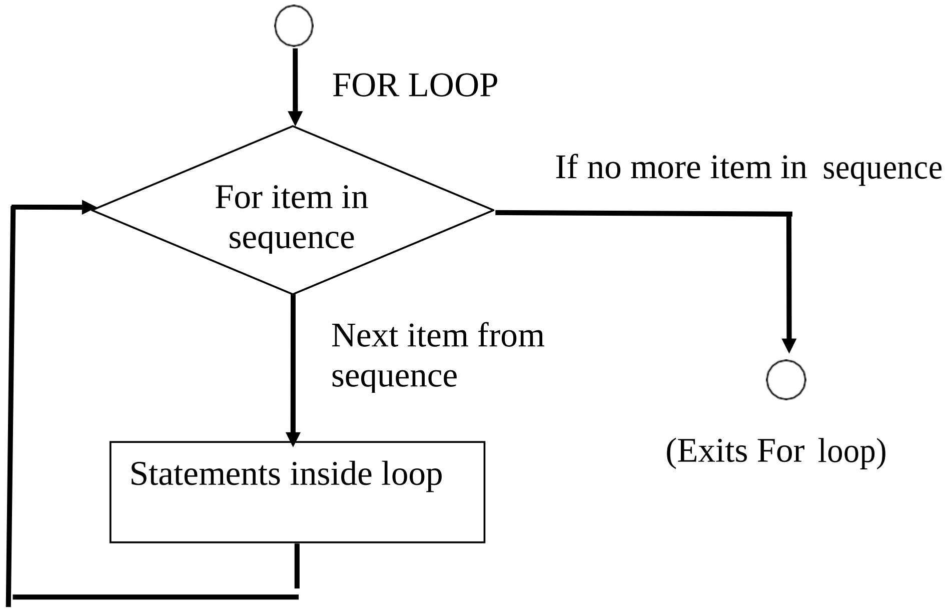
DON'T change any other statements. Run the code. Check if this variant gives the same output as the original program.

What will happen if we use `i < num/2` instead of `i <= num/2` as the while condition? Modify your program, and check what happen.

Chapter II: For-Loop

1. Basic For-loop

`For` loop repeatedly executes the target set of statements for a finite and fixed number of iterations. Python `For` loop can iterate over any sequence



In a new cell, type the following:

```
for n in range(10):
    print('count up: ', n)
print('It is done. Now outside for-loop.')
```

The colon (**:**) is important. It marks the beginning of a block of code

Run the code snippet. You should get the following output:

```
 count up:  0
 count up:  1
 count up:  2
 count up:  3
 count up:  4
 count up:  5
 count up:  6
 count up:  7
 count up:  8
 count up:  9
It is done. Now outside for-loop.
```

In the above example, *n* took on 10 values, 0 to 9. It looks like the value of *n* got incremented by 1 automatically each time the loop was executed. And stopped automatically after 10 values (0 though 9). But that is not an accurate description. Before we proceed further, we need to understand the `range()` function.

2. range() function

`range()` function creates lists containing arithmetic progressions. It is mostly used in *for* loops. The parameters inside the *range*() function must be integers. There are <u>three variants</u> of the *range*() function.

i. `range`(*stopnumber*): Creates a list of numbers from 0 to *stopnumber*−1. Try on the console:

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(1)
[0]
```

ii. `range`(*startnumber*, *stopnumber*): Creates a list of numbers from *startnumber* to *stopnumber*−1, where *startnumber* < *stopnumber*. Try:

```
>>> range(1, 10)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(1, 11)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> range(-10, 0)
[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1]
```

iii. `range`(*startnumber, stopnumber, step_size*): Creates a list of numbers from *startnumber* to at most *stopnumber*−1 and spaced by the *step_size*, where *startnumber* < *stopnumber*. Try:

```
>>> range(1, 11, 2)
[1, 3, 5, 7, 9]
>>> range(1, 11, 3)
[1, 4, 7, 10]
>>> range(0, 100, 10)
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> range(0, -10, -1)
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```

Write the range() function which will output

- [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
- All odd numbers from 3 to 25 (including 25)
- Starting with 7 the first 12 multiples of 7.

3. Back to for-loop

Let's write the code for countdown example that we did using *while* loop.

In a new cell, type:

```
for n in range(10, 0, -1):
   print('countdown: ', n)
print('It is done. Now outside for-loop.')
```

Run the code snippet. The output should be the same as the *while* countdown example (see Section I.1)

The *for*-loop iterates over the values of the list generated by the range() function and in each iteration, it assigns the value to *n*.

Let's write the divisors example using for-loop:

IN a new cell, type:

```
print("To find number of divisors other than 1 & itself")
n = input("Enter any positive integer: ")
num = int(n)
divisors=0
for i in range(2,num):
    if num % i == 0:
        print(i,'is a divisor of', n)
        divisors = divisors+1
print('The number of divisors of', n, 'is', divisors)
```

Run the program multiple times, give various values of integers as input and check the output.

Most of the time we can use either *while* loop or *for* loop, with both giving the same results. There are some instances where one loop is better than the other.

4. Example: Factorial

Let’s write a program to print the factorial of a number.

In a new cell, type the following:

```
print("This program computes n!")
n = input("Enter any positive integer: ")
num = int(n)
fact = 1
for i in range(1, n+1):

    fact = fact*i
print('The factorial of', n, 'is', fact)
```

Run the code snippet. Run the program multiple times, give various values of integers as input and check the output. DOES IT WORK CORRECTLY?

It should not → Find the error and fix it.

Let's write a program to print the first *n* Fibonacci numbers.

```
print('Outputs first 20 numbers in Fibonacci series')
a = 1
b = 1
maxcount = 20
for i in range(1, maxcount+1):
    print(i, 'th number is ', a)
    temp = a + b
    a = b
    b = temp
```

Run the code snippet.

Chapter III: Nested Loops

1. Basic Structure

We can have one loop inside another loop. Basic structure is something as follows:

```
for i in ...
    <outer for-loop statements>
    for j in ...
        <inner for-loop statements>
    <outer for-loop statements>
```

```
while <outer-condition>
    <outer while-loop statements>
    while <inner-condition>
        <inner while-loop statements>
    <outer while-loop statements>
```

```
for i in ...
    <outer for-loop statements>
    while <inner-condition>
        <inner while-loop
        statements>
    <outer for-loop statements>
```

```
while <outer-condition>
    <outer while-loop statements>
    for j in ...
        <inner for-loop statements>
    <outer while-loop statements>
```

2. Example: Numbers Tree Write a program to produce the following output:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
```

In a new cell, type:

```
print("This program print a number tree")
for i in range(1,10):
  for j in range(1,i+1):
      print(j, end = '')
print()
```

The **end = ‘ ’** is to ensure that futher *print* statments are printed on the same line.

The blank *print* statement, prints a new line.

Run the code snippet

Note: The above program can also be written using nested while-loop.

3. Example: Identity Matrix. Write a program to produce the output :

```
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
```

In new cell, type

```
print("This program displays id matrix kind of output")
for i in range(4):
  for j in range(4):
      if j==i:
          print(1, end = '')
      else:
          print(0, end = '')
  print()
```

Run the code snippet.Note: The above program can also be written using nested while-loop.

Else statement with loops: Python supports *else* statement associated with loop statements.

(i) When the *else* statement is used with a for-loop, the else statement is executed when the loop has exhausted iterating the list.

(ii) When the *else* statement is used with a while-loop, the else statement is executed when the condition becomes false.

Loop Control Statements: Loop control changes the execution from its normal sequence. Python has 2 useful control statements:

(i) `break`: Terminates the loop and transfers execution to the statement directly following the loop

(ii) `continue`: Causes the loop to skip the reminder of the loop statements, and immediate retest the condition and reiterate

There is also a third control statement `pass`.

To Do Exercise

In a new notebook, do the following. Use comments / print messages to neatly label the code.

1. Write a code snippet that returns the sum of the squares of even numbers between 2 to *n* where *n* is an input from the user.

2. Write a code snippet**,** that takes as input from the user a positive integer and returns the number of times the number is divisible by 2 before it is less than or equal to 1. For example:

If input is 32; output should be 5. (32 is divisible by 2 five times before it becomes <= 1)

If input is 31, output should be 5 (check and see)

If input is 33, output should be 6.

3. We know that $\sum_{n=0}^{\infty} x^n = \frac{1}{1-x}$ for all $|x| < 1$.

Now, write a code snippet, that takes as input *x* such that $|x| < 1$, and outputs the value of *n* at which the |LHS – RHS| < 0.01 in the above formula.

4. Write a code snippet that takes as input a positive integer from the user, says whether it is a prime number or not.

5. Write a code snippet that prints as output the multiplication tables from 1 to 13, each until 10. Sample output:

```
1 tables: 1 2 3 4 5 6 7 8 9 10
2 tables: 2 4 6 8 10 12 14 16 18 20
3 tables: 3 6 9 12 15 18 21 24 27 30
```

… and so on until 13 tables.

6. Write a code snippet to display all the numbers which are divisible by 11 but not by 2 between 100 and 500. Write as compact a code as possible for this.