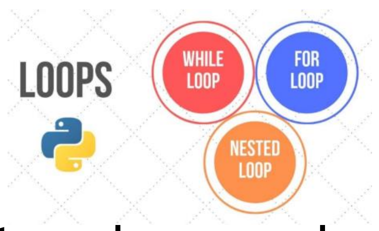


A Closer Look at ***For*** **loops** in Python

One of the *fantastic* things that computer programs do is **repeat procedures over and over and over again** so that you don't have to. This can be done a few ways, but in Python ***For*** *loops* are the most common and versatile method for repeating procedures when solving problems.

A Quick Review:

Most commonly **for** loops are used creating or looping over a sequence of data (a list, characters in a string, integers etc.).

Examples:

For Loops using `range()`

```
for i in range(6):
    print(i)
```

```
Output
0
1
2
3
4
5
```

will loop through code 6 times…0 to 5.

range() is used to control how many times the loop will be repeated.

When working with `range()`, you can pass between 1 and 3 integer ***parameters*** to it:

```
range(start,stop,step)
```

- `start` states the integer value at which the sequence begins. If this is not included then `start` begins at 0
- `stop` is **always required** and is the integer that is counted up to **but not included**
- `step` sets how much to increase (or decrease in the case of negative numbers). If step is omitted then `step` defaults to 1

We'll look at some examples of using the different ***parameters*** in `range()`.

Exercise#1

Using the `range`() function in python create **for** loops that will output each of the following sequences of numbers:

a) 0,1,2,3
b) 1,2,3,4,5,6,7,8,9
c) 2,3,4,5
d) 0,4,8,12,16,20
e) 10,15,20,25,30,35
f) 10,9,8,7,6,5,4,3
g) 1000,975,950,925,900,875

Using the `range()` function in python create for loops that will output each of the following sequences of numbers:

h) 0,1,2,3,4,5,6,7
i) 1,2,3,4,5,6,7
j) 2,3,4,5
k) 0,4,8,12,16,20
l) 10,15,20,25,30,35
m) 10,9,8,7,6,5,4,3
n) 1000,975,950,925,900,875

For Loops using Lists or strings

In python, for loops have been designed to work easily with lists and other data types. *Rather than* looping through a `range()`, you can simply define a list and loop through that list as shown below.

Example:

We'll assign a list to a **variable**, and then loop through the list:

```
sharks = ['hammerhead', 'great white', 'dogfish', 'frilled',
'bullhead', 'requiem']
```

```
for i in sharks:
    print(shark)
```

```
Output
hammerhead
great white
dogfish
frilled
bullhead
requiem
```

You can also use a **`for`** loop to construct a **list** from scratch:

```
numbers = []

for i in range(10):
   numbers.append(i)
print(integers)

Output
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
#{Remember:.append adds stuff to lists.}
```

In the example above, the list numbers is initialized ***empty***, but the for loop "populates" the list.

Exercise#2

Using a `for` loop, `.append(i)`, and `range()` to **create a list** named `up_by_fives[]` that contains a list of numbers that goes up by fives from 0 to 50. Then print out the list.

For loops work great with strings in python too:

```
sammy = 'Sammy'

for letter in sammy:
   print(letter)
```

```
Output:
S
a
m
m
y
```

Exercise#3

Here is a cool **function** that **counts the letters of any word** you give it. **Type it in to Trinket**, test it out, then save it or add it to your library of cool functions.

```
def count_letters(letter,text):
    count = 0
    for c in text:
        if c == letter:
            count = count + 1
    return count

print(count_letters("a","banana"))
```

Exercise#4

Look at each bit of code below. Predict what the does, *then* enter the code into Trinket to see what it does and then save all the examples as exercise#4:

a)

```
count=0
colors = ['red', 'orange', 'yellow', 'green', 'orange']

for x in colors:
  if x == 'orange':
     count = count + 1
print(count)
```

break

b) Example of using **"break"** used in a for loop. A break is used to **end the loop** when needed.

```
fruits = ['apple', 'orange', 'banana', 'cherry']
for x in fruits:
  if x == 'banana':
    break
  print(x)
```

continue

c) Example of using "**continue**" used in a for loop. **Continue** is used to step out of a loop only once and then continue when needed.

"continue" will ***skip the block of code under certain conditions***

```
fruits = ['apple', 'orange', 'banana', 'cherry']
for x in fruits:
  if x == 'banana':
    continue
  print(x)
```

Exercise#5

Nested loops

A "Nested" for loop is simply a **for** loop *inside* another **for** loop.

In the example below each *store owner* gets printed with a second loop that will print *each* fruit with *each* owner. See if you can predict the output for this code...then put it into Trinket and save it as part of **exercise#5**.

```
Store_owner = ['Tim','Sandy','Bill']
Fruits = ['apple','bananas','cherries']

for x in Store_owner:
  for y in fruits:
    print(x, y)
```

Below is another "**Nested**" for loop (a **For** loop inside another **for** loop).
In the example below **'Hey'** gets printed **once** during the first pass of the loops, then twice on the 2nd pass through the loops, then 3 times on the 3rd pass through loops etc... See if you can predict the output for this code will look like...then put it into Trinket and save it as part of **exercise#5**.

```
for i in range(7):
    for j in range(i):
        print('Hey')
    print('')
```

See if you can predict the output for the following 2 blocks of code will look like...then put it into Trinket and save it as part of **exercise#5**.

```
persons = [ "John", "Marissa", "Pete", "Dayton" ]
restaurants = [ "Japanese", "American", "Mexican", "French" ]

for person in persons:
    for restaurant in restaurants:
        print(person + " eats " + restaurant)
```

```
for i in range( 1, 6 ):
    for j in range(i):
        print ('*'),    # the comma is important! It lets you print on the same line!
    print ('')
```

Your turn:

Exercise #6

Use a **nested for loop** to **print out** a display that clearly shows that each person's backpack contains *each* of the following items so the group is ready for the hike:

```
backpacks=['john_bp', 'stacy_bp', 'cory_bp']
items=['socks', 'matches', 'water', 'food']
```

Exercise #6

Use a **nested for loop** to **print out** the following display

```
*
**
***
****
*****
```

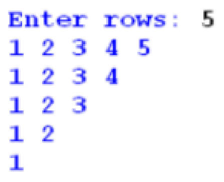
Exercise #7

Use a **nested for loop** to **print out** the following display

```
1
22
333
4444
55555
666666
```

Exercise #7

Create program that will take a integer between 3 and 10 (from the user) as input and use a **nested for loop** to **print out** the following inverted right angle triangle display



```
Enter rows: 5
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

Exercise #8

Use a *single* for loop to print all the numbers from 0 to 15 *except* 3, 6, and 9. Use the 'continue' command.
Expected Output : 0 1 2 4 5 7 8 10 11 12 13 14 15

Exercise #9

Write a program containing a **nested for loop** that will allow the user to input 2 numbers: *Columns* and *Rows*. Then the program will output a **table of zeros** based on the input given. Example:

```
6 and 4
000000
000000
000000
000000
```

Exercise #10

Use a **nested for loop** to create the black and white checker board shown.
You *might* have to use an *if* statement.

```
black white black white black
white black white black white
black white black white black
white black white black white
black white black white black
```

Bonus: Times Tables

Use a **nested for loop** to create *clearly organized* multiplication table when give integer by the user - example: 5

```
1  2  3  4  5
2  4  6  8  10
3  6  9  12 15
4  8  12 16 20
5  10 15 20 25
```