Standard For loop and List Comprehension

In [6]:
```
#Syntax
# for val in sequence:
#       Body of for
```

In [10]:
```
h_letters = []

for letter in 'human':
    h_letters.append(letter)

print(h_letters)
```

```
['h', 'u', 'm', 'a', 'n']
```

List Comprehension

List comprehensions are used for creating new lists from other iterables. As list comprehensions return lists, they consist of brackets containing the expression, which is executed for each element along with the for loop to iterate over each element.

In [11]:
```
#Syntax
# my_list = [expression for item in list]
```

In [12]:
```
h_letters = [ letter for letter in 'human' ]
print( h_letters)
```

```
['h', 'u', 'm', 'a', 'n']
```

Matrix Addition using Nested Loop.

In [13]:
```
X = [[12,7,3],
    [4 ,5,6],
    [7 ,8,9]]

Y = [[5,8,1],
    [6,7,3],
    [4,5,9]]

result = [[0,0,0],
          [0,0,0],
          [0,0,0]]

# Outer loop if to iterate through rows
for i in range(len(X)):
   # Innner loop is to iterate through columns
   for j in range(len(X[0])):
       result[i][j] = X[i][j] + Y[i][j]

for r in result:
   print(r)
```

```
[17, 15, 4]
[10, 12, 9]
[11, 13, 18]
```

Matrix Addition using Nested List Comprehension

In [14]:
```
X = [[12,7,3],
    [4 ,5,6],
    [7 ,8,9]]

Y = [[5,8,1],
    [6,7,3],
    [4,5,9]]

result = [[X[i][j] + Y[i][j]  for j in range(len(X[0]))] for i in ran
ge(len(X))]

for r in result:
   print(r)
```

```
[17, 15, 4]
[10, 12, 9]
[11, 13, 18]
```

Note: The main difference is in appearance and runtime speed. List comprehension is shorter, easier to understand and faster in execution time.