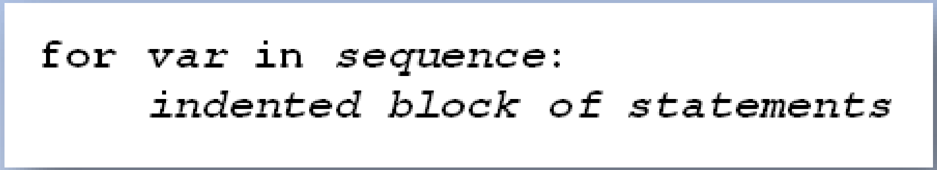
## The *for* Loop

- Used to iterate through a sequence of values
- General form of a for loop



```
for var in sequence:
    indented block of statements
```

- Sequence can be
  - Arithmetic progression of numbers
  - String
  - List
  - File object

## The *for* Loop

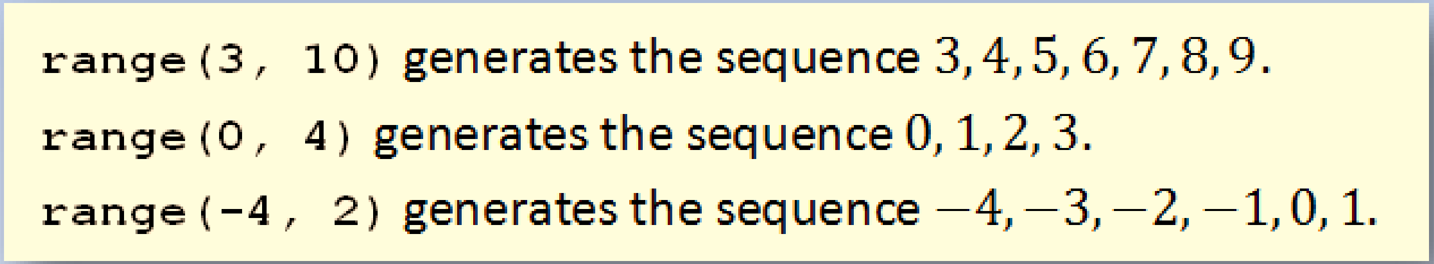
- Variable is successively assigned each value in the sequence

```
for var in sequence:
    indented block of statements
```

- Indented block of statements executed after each assignment
- Physical indentation tells interpreter where block starts and stops

# Looping Through Arithmetic Progression of Numbers

- Range function is used to generate an arithmetic progression



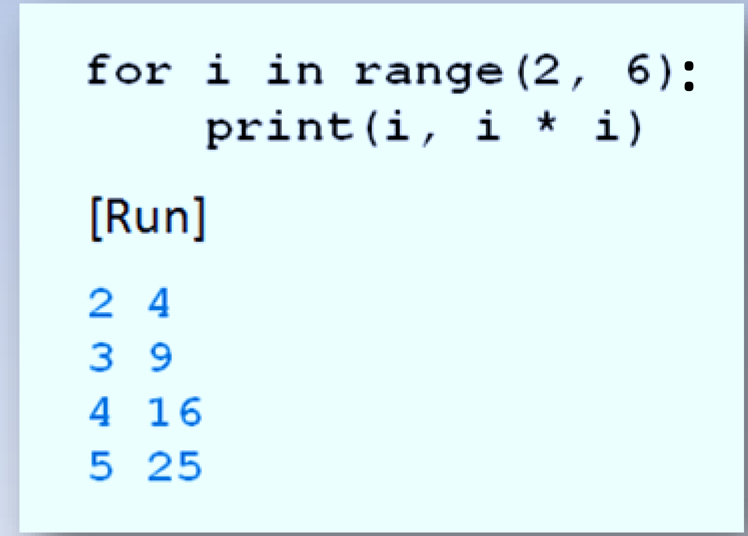
`range(3, 10)` generates the sequence $3, 4, 5, 6, 7, 8, 9$.

`range(0, 4)` generates the sequence $0, 1, 2, 3$.

`range(-4, 2)` generates the sequence $-4, -3, -2, -1, 0, 1$.

# Looping Through Arithmetic Progression of Numbers

- Example: Code displays four integers and their squares



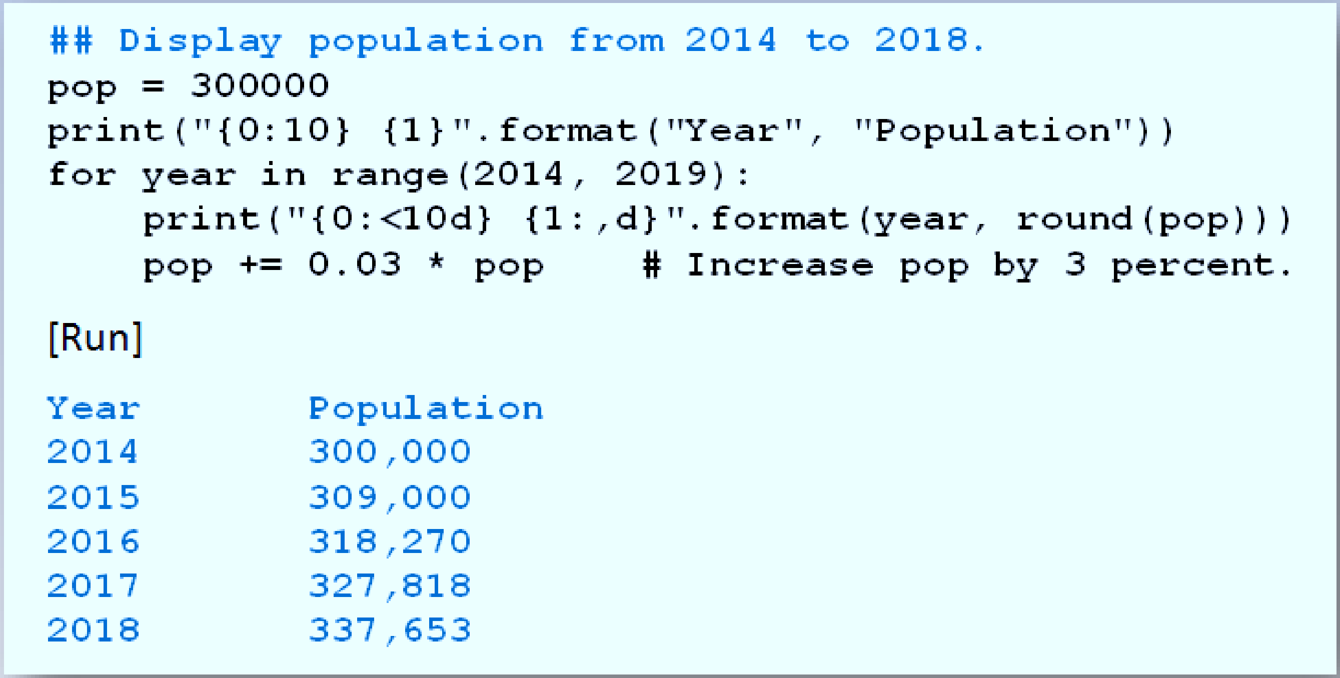
```
for i in range(2, 6):
    print(i, i * i)
```

[Run]

```
2 4
3 9
4 16
5 25
```

# Looping Through Arithmetic Progression of Numbers

- Example: Program displays a table showing the population each year until 2018 increasing by 3 percent every year

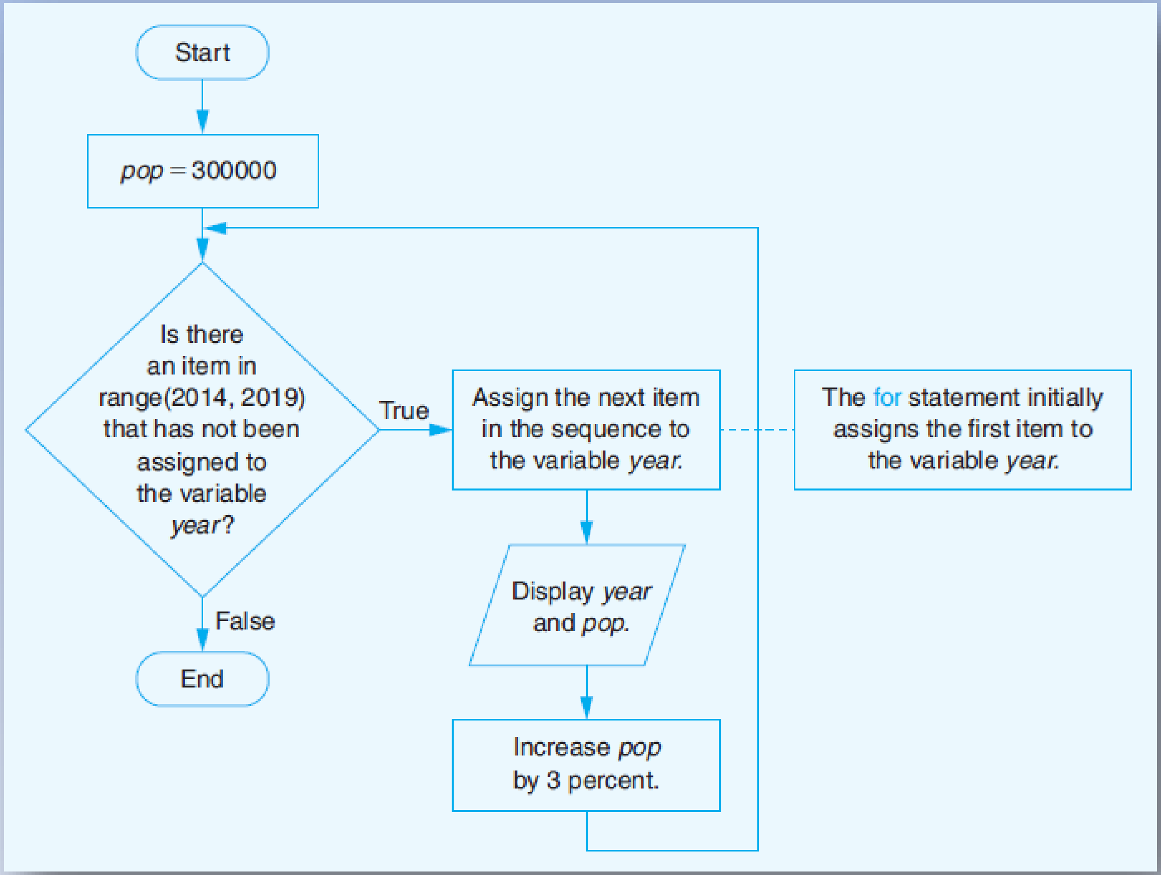


```
## Display population from 2014 to 2018.
pop = 300000
print("{0:10} {1}".format("Year", "Population"))
for year in range(2014, 2019):
    print("{0:<10d} {1:,d}".format(year, round(pop)))
    pop += 0.03 * pop     # Increase pop by 3 percent.
```

[Run]

```
Year       Population
2014       300,000
2015       309,000
2016       318,270
2017       327,818
2018       337,653
```

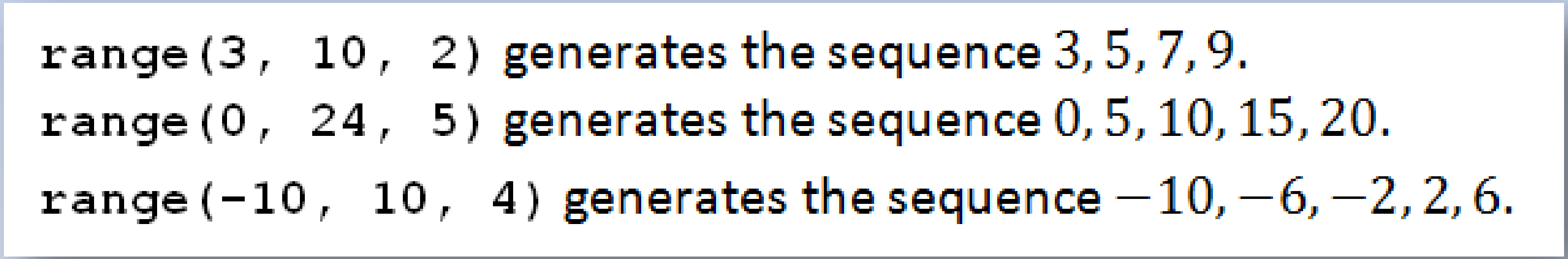
# Looping Through Arithmetic Progression of Numbers


Start
pop = 300000
Is there an item in range(2014, 2019) that has not been assigned to the variable year?
True
False
End
Assign the next item in the sequence to the variable year.
The for statement initially assigns the first item to the variable year.
Display year and pop.
Increase pop by 3 percent.

Flowchart for Population Example

## Step Values for the *range* Function

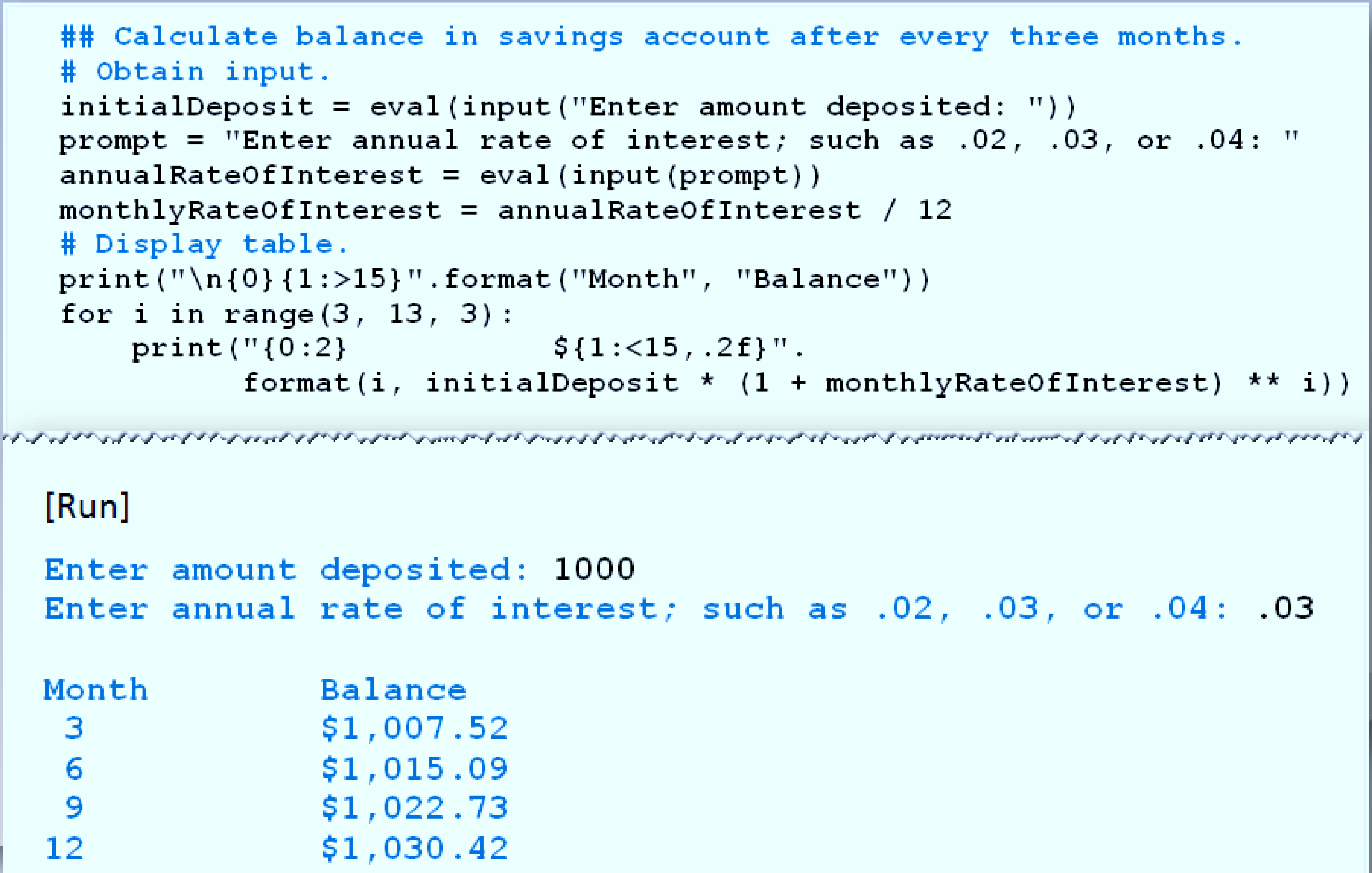
- Variation of the range function generates a sequence of integers
  - Successive integers differ by a value other than 1
- Examples



`range(3, 10, 2)` generates the sequence $3, 5, 7, 9$.
`range(0, 24, 5)` generates the sequence $0, 5, 10, 15, 20$.
`range(-10, 10, 4)` generates the sequence $-10, -6, -2, 2, 6$.

# Step Values for the *range* Function

- Example: Program requests: amount deposited; annual rate of interest
  - then calculates balance after each quarter-year for four quarters.



```
## Calculate balance in savings account after every three months.
# Obtain input.
initialDeposit = eval(input("Enter amount deposited: "))
prompt = "Enter annual rate of interest; such as .02, .03, or .04: "
annualRateOfInterest = eval(input(prompt))
monthlyRateOfInterest = annualRateOfInterest / 12
# Display table.
print("\n{0}{1:>15}".format("Month", "Balance"))
for i in range(3, 13, 3):
    print("{0:2}           ${1:<15,.2f}".
          format(i, initialDeposit * (1 + monthlyRateOfInterest) ** i))
```

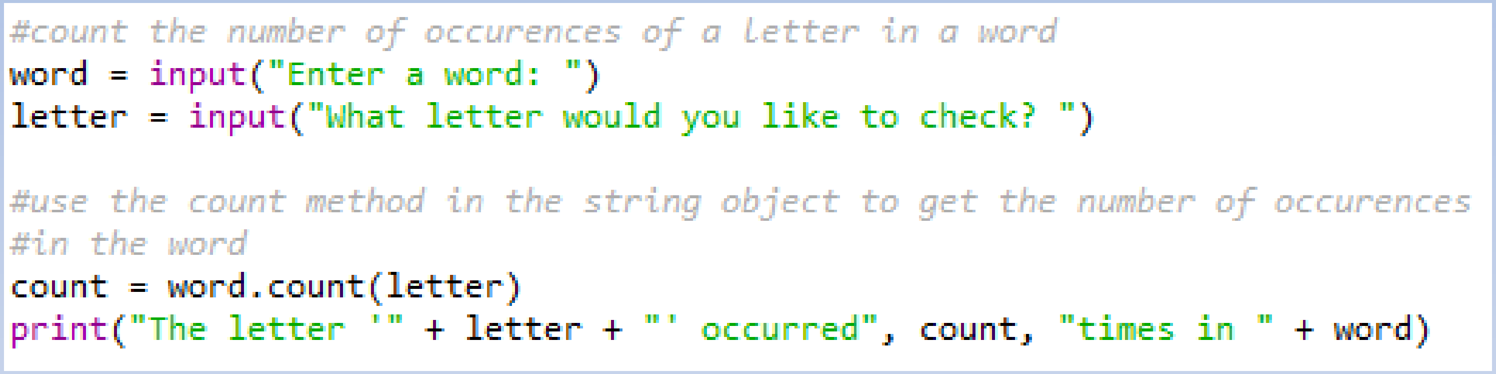
[Run]

```
Enter amount deposited: 1000
Enter annual rate of interest; such as .02, .03, or .04: .03

Month        Balance
 3           $1,007.52
 6           $1,015.09
 9           $1,022.73
12           $1,030.42
```

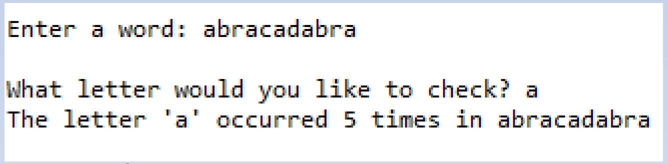
# Looping Through Characters of a String

- Example: Counting the number of occurrences of a character in a string



```python
#count the number of occurences of a letter in a word
word = input("Enter a word: ")
letter = input("What letter would you like to check? ")

#use the count method in the string object to get the number of occurences
#in the word
count = word.count(letter)
print("The letter '" + letter + "' occurred", count, "times in " + word)
```



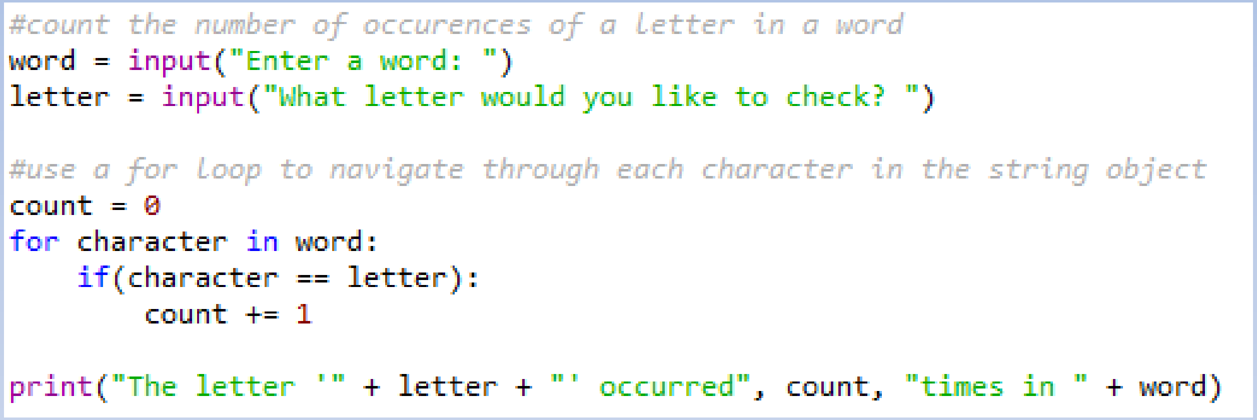
```
Enter a word: abracadabra

What letter would you like to check? a
The letter 'a' occurred 5 times in abracadabra
```

- Can we do the same without using *count* method in the *string* object?

# Looping Through Characters of a String

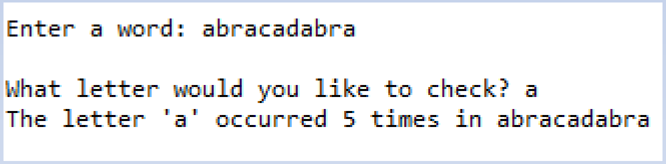
- Example: Counting the number of occurrences of a character in a string *using a for loop*



```python
#count the number of occurences of a letter in a word
word = input("Enter a word: ")
letter = input("What letter would you like to check? ")

#use a for loop to navigate through each character in the string object
count = 0
for character in word:
    if(character == letter):
        count += 1

print("The letter '" + letter + "' occurred", count, "times in " + word)
```



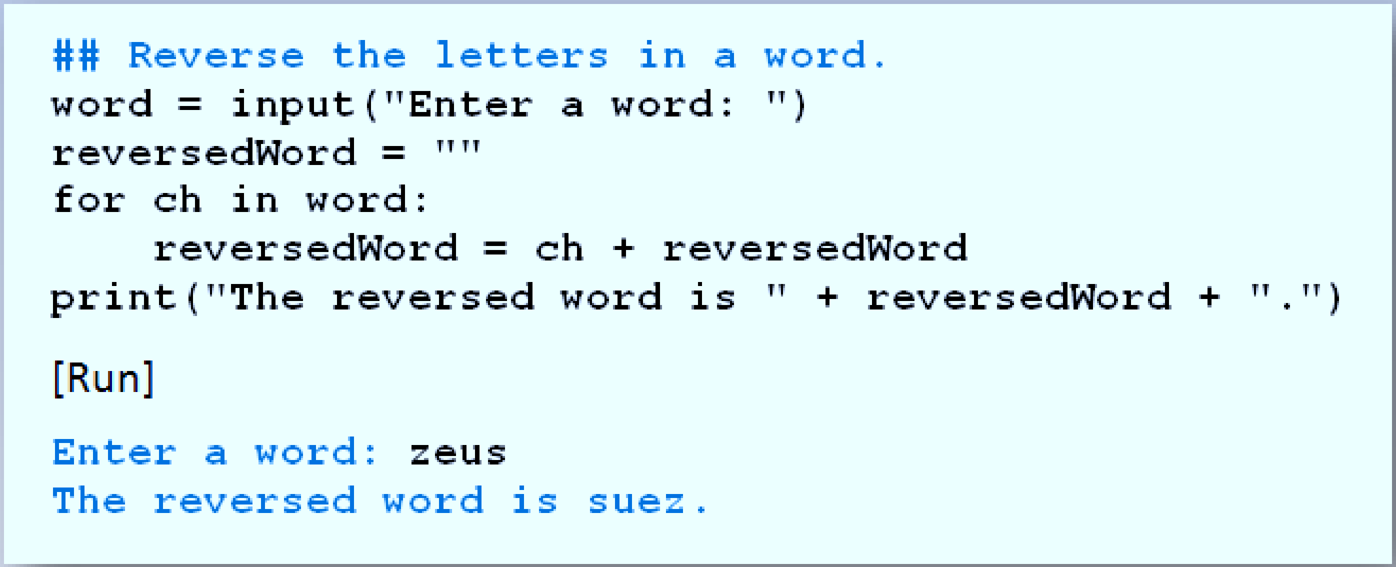
```
Enter a word: abracadabra

What letter would you like to check? a
The letter 'a' occurred 5 times in abracadabra
```

- The for statement is designed to allow you to iterate over the elements of a sequence or other **iterable** object. ***Strings** in **Python** are **iterable***.

# Looping Through Characters of a String

- Example: Program requests a word as input and displays it backwards.



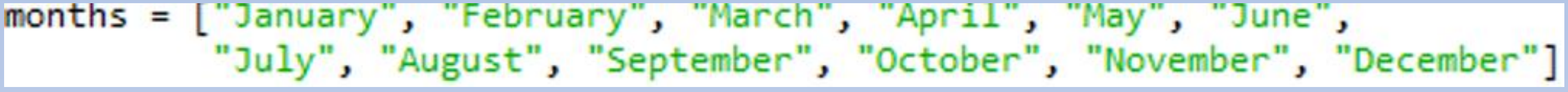
```python
## Reverse the letters in a word.
word = input("Enter a word: ")
reversedWord = ""
for ch in word:
    reversedWord = ch + reversedWord
print("The reversed word is " + reversedWord + ".")
```

[Run]

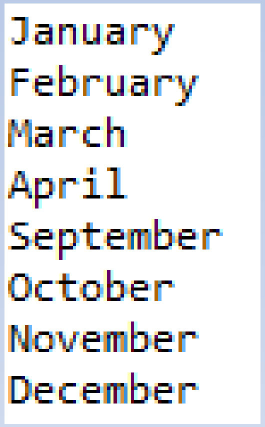
```
Enter a word: zeus
The reversed word is suez.
```

# Looping Through Items of a List

- Example: Program displays the months whose names contains the letter *r*



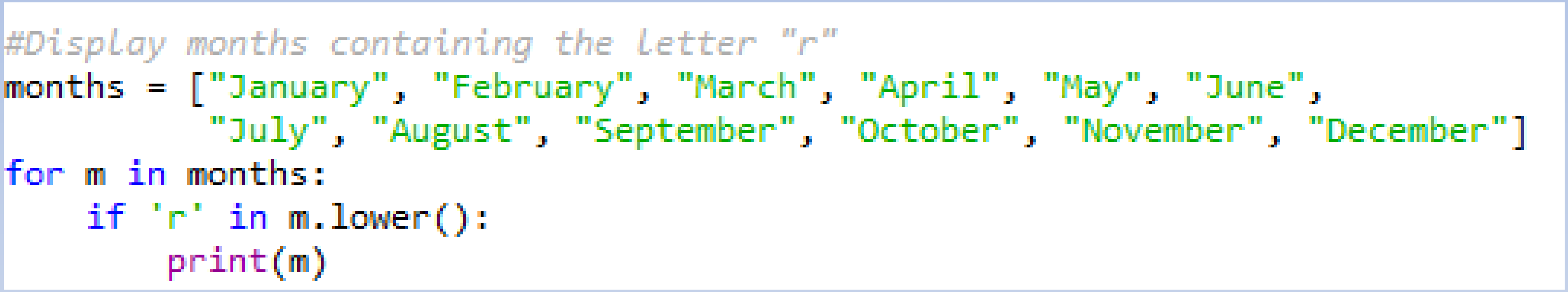
```
months = ["January", "February", "March", "April", "May", "June",
          "July", "August", "September", "October", "November", "December"]
```



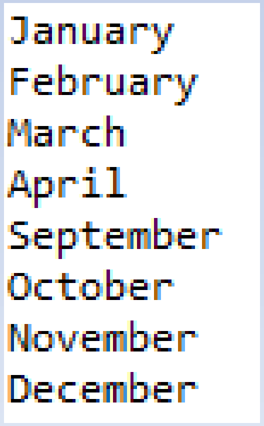
```
January
February
March
April
September
October
November
December
```

## Looping Through Items of a List

- Example: Program displays the months whose names contains the letter *r.*



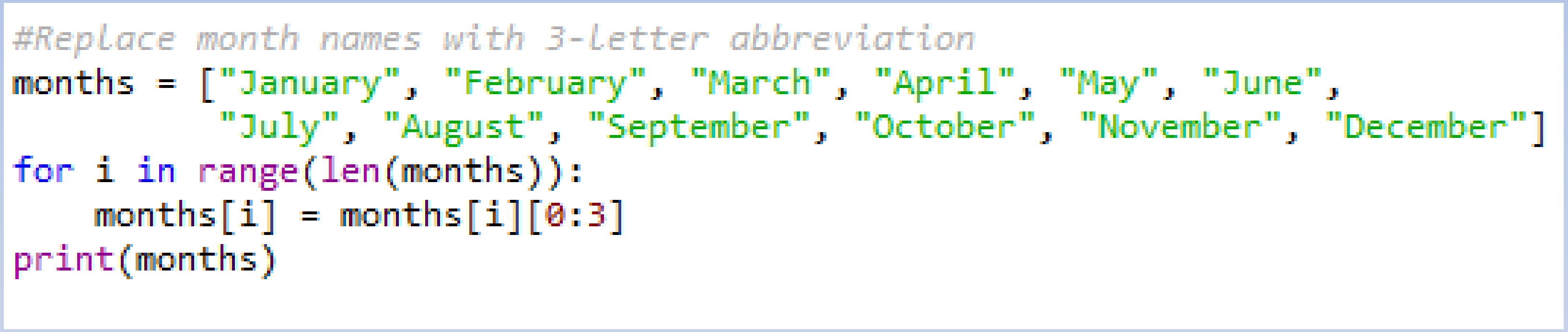
```
#Display months containing the letter "r"
months = ["January", "February", "March", "April", "May", "June",
          "July", "August", "September", "October", "November", "December"]
for m in months:
    if 'r' in m.lower():
        print(m)
```



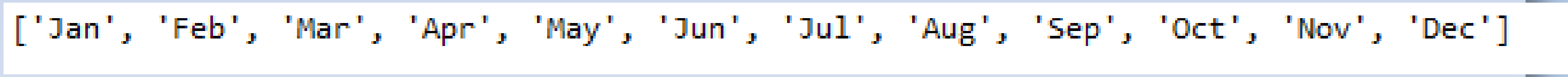
```
January
February
March
April
September
October
November
December
```

# Looping Through Items of a List

- Example: Program replaces the name of each month with its three-letter abbreviation.



```python
#Replace month names with 3-letter abbreviation
months = ["January", "February", "March", "April", "May", "June",
          "July", "August", "September", "October", "November", "December"]
for i in range(len(months)):
    months[i] = months[i][0:3]
print(months)
```



```
['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
```

## Boolean- and List-valued Functions

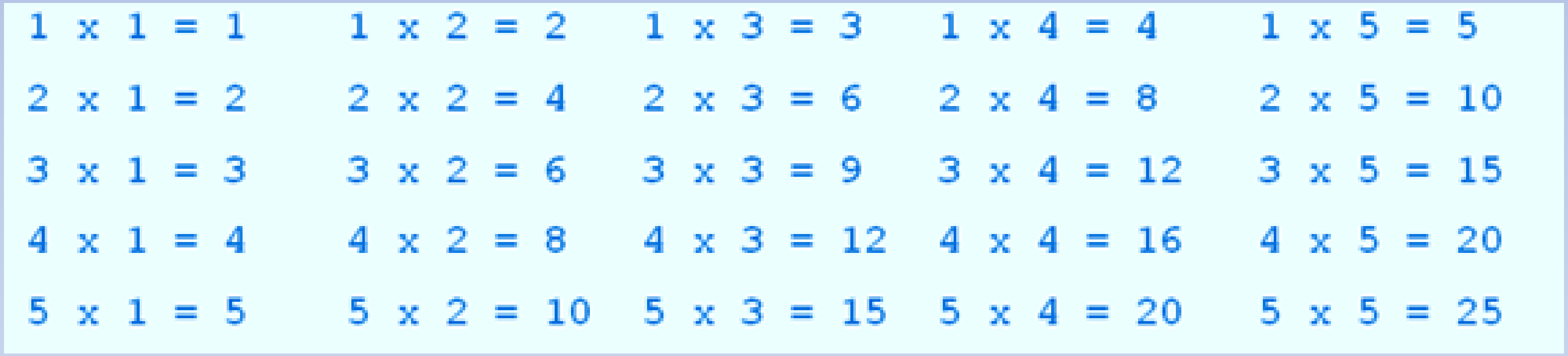
- Case Study:
  - Write a function to check if a word has all the vowels in it (return a Boolean value)
  - Write a function to return all the vowels in it as a list
  - Create a separate module to store the functions
  - The “main” function should be in a different file which imports the module containing the functions needed

## Nested *for* Loops

- Body of for loop can contain any type of Python statement
  – Can contain another for loop.
- Second loop must be completely contained inside the first loop
  – Must have a different loop variable

## Nested *for* Loops

- Example: Program displays a multiplication table for the integers from 1 to 5



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

# Nested *for* Loops

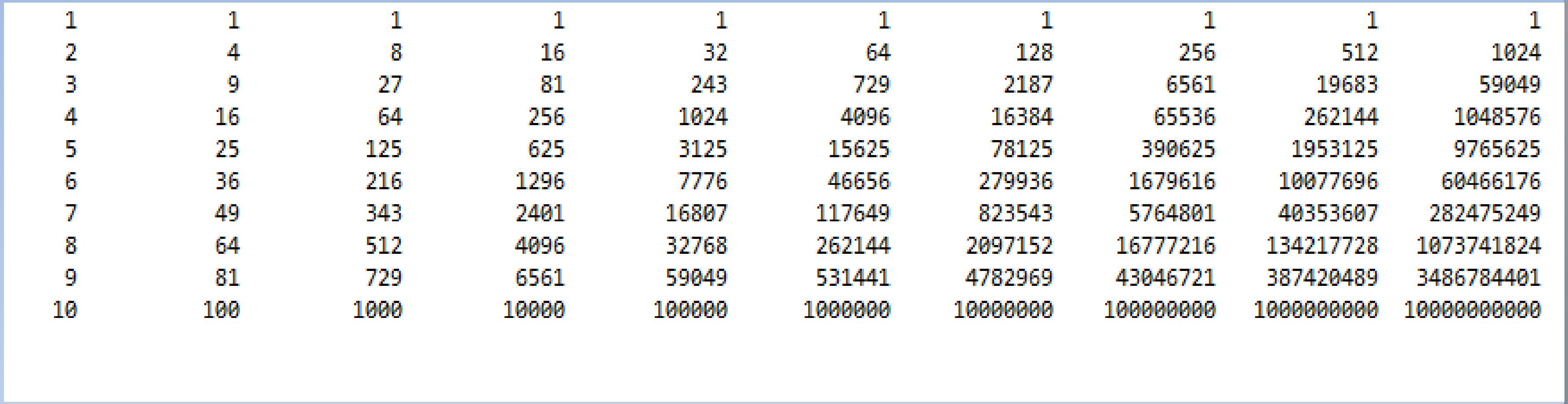
- Example: Program displays a multiplication table for the integers from 1 to 5

```
## Display a multiplication table for the numbers from 1 through 5.
for m in range(1, 6):
    for n in range(1, 6):
        print(m, 'x', n, '=', m * n, "\t", end="")
    print()
```

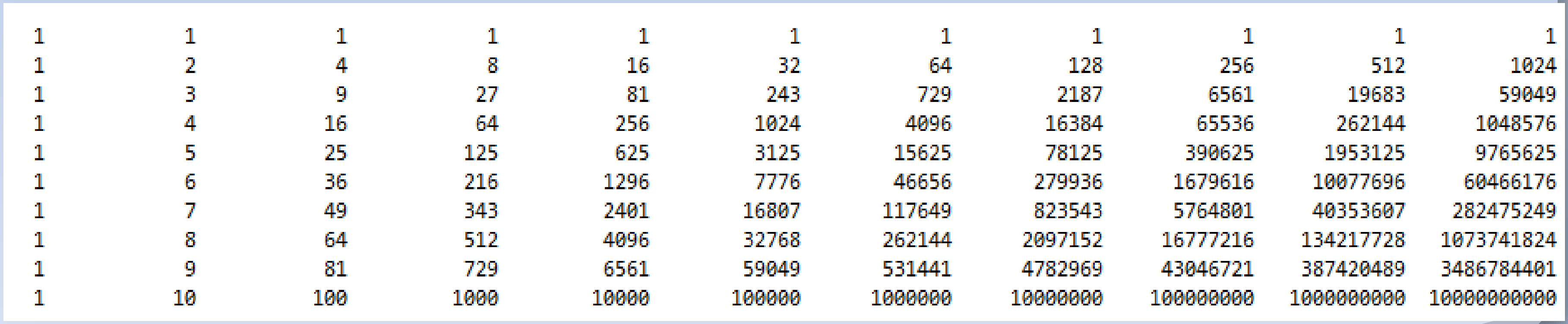
[Run]

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

# Nested *for* Loops?

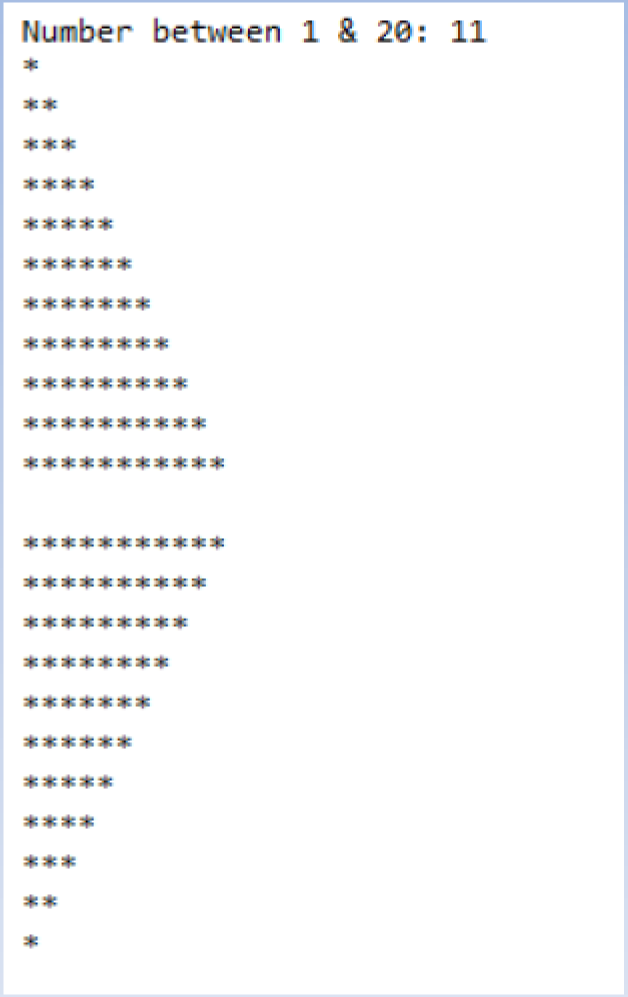


```
           1           1           1           1           1           1           1           1           1           1
           2           4           8          16          32          64         128         256         512        1024
           3           9          27          81         243         729        2187        6561       19683       59049
           4          16          64         256        1024        4096       16384       65536      262144     1048576
           5          25         125         625        3125       15625       78125      390625     1953125     9765625
           6          36         216        1296        7776       46656      279936     1679616    10077696    60466176
           7          49         343        2401       16807      117649      823543     5764801    40353607   282475249
           8          64         512        4096       32768      262144     2097152    16777216   134217728  1073741824
           9          81         729        6561       59049      531441     4782969    43046721   387420489  3486784401
          10         100        1000       10000      100000     1000000    10000000   100000000  1000000000 10000000000
```



```
1           1           1           1           1           1           1           1           1           1           1
1           2           4           8          16          32          64         128         256         512        1024
1           3           9          27          81         243         729        2187        6561       19683       59049
1           4          16          64         256        1024        4096       16384       65536      262144     1048576
1           5          25         125         625        3125       15625       78125      390625     1953125     9765625
1           6          36         216        1296        7776       46656      279936     1679616    10077696    60466176
1           7          49         343        2401       16807      117649      823543     5764801    40353607   282475249
1           8          64         512        4096       32768      262144     2097152    16777216   134217728  1073741824
1           9          81         729        6561       59049      531441     4782969    43046721   387420489  3486784401
1          10         100        1000       10000      100000     1000000    10000000   100000000  1000000000 10000000000
```

# Nested *for* Loops?



```
Number between 1 & 20: 11
*
**
***
****
*****
******
*******
********
*********
**********
***********

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