Lists, For Loops, and List Comprehensions

May 17, 2022

1 Lists

A list is a sequence of values which can be of any type. The values in a list are called elements or items. A list is enclosed in square brackets with the values separated by commas.

There are several ways to create a list. A simple way is to enclose the elements in brackets per:

```
data = [10, 23, 45, 67]
planets = ['earth', 'mercury', 'mars']
```

The first example is a list of four integers assigned to the variable `data`. The second is a list of three strings assigned to the variable `planets`. The elements of a list don't have to be the same type. Lists can also be nested in other lists. The following list contains a string, a float, an integer, and another list:

```
mixed_list = ['earth', 2.1, 5, [15.1, 23.5, 67.5]]
```

An empty list contains no elements and can be created with empty brackets `[]`.

1.1 Traversing a list

The most common way to traverse the elements of a list is with a `for` loop using the standard syntax:

```
for planet in planets:
    print(planet)
```

This works well to read the elements of the list. If you want to write or update the elements, you need the indices. A common way to do that is to combine the built-in functions range and len:

```
for i in range(len(numbers)):
    numbers[i] = numbers[i] * 2
```

This loop traverses the list and updates each element. `len` returns the number of elements in the list. `range` returns a list of indices from 0 to n−1, where n is the length of the list. Each time through the loop `i` gets the index of the next element. The assignment statement in the body uses `i` to read the old element value and assign the new value.

1.2 List operations

The `+` operator concatenates lists:

```
a = [1, 2, 3]
b = [4, 5, 6]
```

```
c = a + b
print(c)
```

```
[1, 2, 3, 4, 5, 6]
```

The * operator repeats a list a given number of times:

```
>>> [0] * 4
[0, 0, 0, 0]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

The first example repeats `[0]` four times. The second example repeats the list `[1, 2, 3]` three times.

1.3 List methods

Python provides methods that operate on lists. The `append` method adds a new element to the end of a list:

```
>>> t = ['a', 'b', 'c']
>>> t.append('d')
>>> t
['a', 'b', 'c', 'd']
```

The `sort` methods arranges the elements of the list from low to high:

```
>>> t = ['d', 'c', 'e', 'b', 'a']
>>> t.sort()
>>> t
['a', 'b', 'c', 'd', 'e']
```

2 For Loop

A `for` loop is used for iterating over a sequence (either a list, a tuple, a dictionary, a set, or a string). The `for` loop has this syntax where the nested statements will be repeated for each item in the sequence:

```
for variable in sequence:
    one or more statements
```

One way to control the loop is to use a `list` containing a sequence in brackets separated by commas:

```
for word in ["one", "two", "three"]:
    print(word)
```

Another way is to use the `range` function which provides a sequence of numbers and is commonly used with `for` loops to specify the number of loops to run. It is specified as `range(start, stop)` and returns a sequence of integers from *start* up to, but not including the *stop*. The *start* parameter is optional and defaults to zero.

The following will print the numbers 1 through 10, each on a separate line.

```
for number in range(1, 11):
    print(number)
```

Below are examples of using `for` loops on strings, lists of strings, and lists of numbers as the iterables.

[1]:
```
for character in 'Hello SE4003':
    print(character)

# a list of strings that can be iterated on
se4003_students = [
"Auld, Sean",
"Camp, Daniel",
"Daley, Steven",
"Dogum, Gregory",
"Gizas, Ashley",
"Gonzalez, Victor",
"Hill, Benjamin",
"Hogan, Martin",
"Iya, Gabriel",
"Jones Maia, Kristin",
"Kylander, Paul",
"Meisner, Megan",
"Meszaros, Michele",
"Novoa, Jonathan",
"Patterson, Dawn",
"Stokes, Joshua",
"Vermeulen, Suzanne",
"Vey, Nathan",
"Villarreal, Rene",
"Willis, Jerald",
"Wilson, Nicole",
]

# loop through list and assign each value to the variable 'name' in each␣
 ↪iteration
for name in se4003_students:
  print ('Hello', name)

data_list = [22, 45, 7.5, 1.8]

for value in data_list:
  print (value)
```

```
H
e
l
l
```

```
o

S
E
4
0
0
3
Hello Auld, Sean
Hello Camp, Daniel
Hello Daley, Steven
Hello Dogum, Gregory
Hello Gizas, Ashley
Hello Gonzalez, Victor
Hello Hill, Benjamin
Hello Hogan, Martin
Hello Iya, Gabriel
Hello Jones Maia, Kristin
Hello Kylander, Paul
Hello Meisner, Megan
Hello Meszaros, Michele
Hello Novoa, Jonathan
Hello Patterson, Dawn
Hello Stokes, Joshua
Hello Vermeulen, Suzanne
Hello Vey, Nathan
Hello Villarreal, Rene
Hello Willis, Jerald
Hello Wilson, Nicole
22
45
7.5
1.8
```

2.1 List Comprehensions

Below is an example of using a `for` loop to populate a list with random values that will be reduced to a single line in a "list comprehension".

```
import random
random_values = []
for num in range(10):
    random_values.append(random.random())
```

The above can be expressed in a single line to populate the list enclosed in brackets:

```
random_values = [random.random() for num in range(10)]
```

The list comprehension statement can also be printed directly to display the values in a single line as below.

[4]:
```
print([random.random() for num in range(10)])
```

```
[0.04779021240901815, 0.4363164510910684, 0.2864155034722283,
0.6737370047157383, 0.966817485819037, 0.1991951725147978, 0.9736462899158717,
0.21746369981908975, 0.7404519674402908, 0.6368308995531794]
```

List comprehensions can combine multiple for loops for nested logic such as below. During execution it sets the outer index to its initial value, iterates through the inner loop, then goes back to the outer loop next value and continues.

[4]:
```
print([inner*outer for inner in [1, 2, 3, 4] for outer in [100, 200, 300]])
```

```
[100, 200, 300, 200, 400, 600, 300, 600, 900, 400, 800, 1200]
```

3 Example Program

The Monte Carlo simulation program below demonstrates using lists and a `for` loop to control the simulation runs.

[3]:
```
# demonstrate monte carlo simulation with time loop to calculate position over␣
 ↪time and calculate simple statistics

import random
import statistics # used for statistics

dt = .25 # Timestep
num_iterations = 50
verbose = False # False will turn off single run intermediate output

car1_distance, car2_distance = [], [] # initialize output measure lists

for iteration in range(num_iterations):
  time = 0.
  position1 = 50 # Initial position (miles)
  position2 = 50 # Initial position (miles)
  # print header for time output and initial states
  if verbose: print ("Time  Position 1  Position 2")
  if verbose: print ("%6.2f %6.2f %6.2f" % (time, position1, position2))
  while time < 10: # Run a 10 hour race
      time = time + dt
      #random velocities
      velocity1 = random.randint(35, 65)
      velocity2 = random.randint(30, 60)
      position1 = position1 + velocity1*dt
      position2 = position2 + velocity2*dt
      if verbose: print ("%6.2f %6.2f %6.2f" % (time, position1, position2))

  car1_distance.append(position1)
```

```
  car2_distance.append(position2)

  if verbose: print ("Final position: %6.2f" % position1)
  if verbose: print ("Final position2: %6.2f" % position2)

print (f"Car 1 Distance Mean  = {statistics.mean(car1_distance) :3.0f} Miles")
print (f"Car 2 Distance Mean  = {statistics.mean(car2_distance) :3.0f} Miles")

print(car1_distance)
print(car2_distance)
```

```
Car 1 Distance Mean  = 548 Miles
Car 2 Distance Mean  = 503 Miles
[539.5, 560.5, 547.25, 582.75, 542.5, 577.75, 540.75, 559.5, 534.5, 535.5,
554.75, 554.5, 556.75, 545.0, 535.0, 540.0, 546.25, 521.75, 556.5, 548.75,
541.0, 544.25, 560.75, 557.5, 562.75, 558.75, 535.5, 538.75, 542.0, 530.0,
551.25, 564.0, 557.5, 548.25, 540.75, 547.0, 531.75, 538.0, 546.75, 547.0,
559.25, 525.25, 545.0, 559.75, 543.25, 558.5, 565.5, 537.5, 542.25, 556.5]
[502.0, 522.5, 481.5, 472.0, 508.75, 507.25, 482.0, 486.25, 493.25, 504.25,
504.5, 531.5, 514.75, 502.25, 511.0, 501.0, 497.25, 505.75, 498.0, 502.75,
516.0, 500.75, 506.5, 503.25, 510.0, 498.5, 518.0, 517.5, 499.25, 498.75, 490.5,
509.5, 494.75, 493.0, 512.0, 495.75, 488.0, 510.5, 511.75, 521.75, 507.5,
536.25, 492.75, 505.5, 504.25, 491.25, 491.0, 523.75, 511.5, 484.75]
```