16. More On Lists

Topics:
References
Alias
More on Slicing
Merging Sorted Lists
Comparing Lists

You can use `==` to compare two lists

```python
>>> x = [10,20,30,40]
>>> y = [10,20,30,40]
>>> x==y
True
```
Comparing Lists

You can use `==` to compare two lists.

The Boolean expression $x == y$ is True because $x$ and $y$ have the same length and identical values in each element.
Comparing Lists

You can use == to compare two lists

```python
>>> x = [1, 2, 3]
>>> y = [1.0, 2.0, 3.0]
>>> x==y
True
```

If there are ints and floats, convert everything to float then compare
Comparing Lists

Do not use <, <=, >, >= to compare two lists

```python
>>> x = [10, 20, 30, 40]
>>> y = [11, 21, 31, 41]
>>> x<y
True
>>> y<x
True
```

Unpredictable
Aliasing

This:

\[ x = [10, 20, 30, 40] \]
\[ y = x \]

Results in this:

\[ x \rightarrow 0 \quad \rightarrow 10 \]
\[ 1 \rightarrow 20 \]
\[ 2 \rightarrow 30 \]
\[ 3 \rightarrow 40 \]
Things to say:

x and y are variables that refer to the same list object.

The object is aliased because it has more than one name.
Tracking Changes

\[ x = [10, 20, 30, 40] \]
\[ y = x \]
\[ y = [1, 2, 3] \]
Tracking Changes

\[ x = [10, 20, 30, 40] \]
\[ y = x \]
\[ y = [1, 2, 3] \]

\[ x \rightarrow \begin{array}{c|c}
0 & 10 \\
1 & 20 \\
2 & 30 \\
3 & 40 \\
\end{array} \]

\[ y \rightarrow \begin{array}{c|c}
0 & 10 \\
1 & 20 \\
2 & 30 \\
3 & 40 \\
\end{array} \]
Tracking Changes

\[ x = [10, 20, 30, 40] \]
\[ y = x \]
\[ y = [1, 2, 3] \]
The is Operator

```python
>>> x = [10, 20, 30, 40]
>>> y = [10, 20, 30, 40]
>>> x is y
False
```

Even though the two lists have the same component values, `x` and `y` do not refer to the same object.
The is Operator

```python
>>> x = [10,20,30,40]
>>> y = x
>>> x is y
True
```

```
0 --> 10
1 --> 20
2 --> 30
3 --> 40
```

x and y refer to the same object
Making a Copy of a List

\[ x = [10, 20, 30, 40] \]
\[ y = \text{list}(x) \]
Making a Copy of a List

\[ x = [10,20,30,40] \]

\[ y = \text{list}(x) \]
Slices Create new Objects

\[
x = [10, 20, 30, 40] \\
y = x[1:] \\
x --> \\
0 --> 10 \\
1 --> 20 \\
2 --> 30 \\
3 --> 40
\]
Slices Create New Objects

\[
x = [10, 20, 30, 40]
\]

\[
y = x[1::]
\]
Careful!

\[
x = [10, 20, 30, 40]
\]

\[
y = x
\]

\[
y = x[1:]
\]
Careful!

\[
x = [10, 20, 30, 40]
\]

\[
y = x
\]

\[
y = x[1:]
\]
Careful!

\[
x = [10, 20, 30, 40] \\
y = x \\
y = x[1:] \\
\]

\[
\begin{array}{c|c}
0 & 10 \\
1 & 20 \\
2 & 30 \\
3 & 40 \\
\end{array}
\]

\[
\begin{array}{c|c}
0 & 20 \\
1 & 30 \\
2 & 40 \\
\end{array}
\]
Void Functions

\[ x = [40, 20, 10, 30] \]
\[ y = x.sort() \]

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
</tr>
</tbody>
</table>

\[ x \rightarrow \]
\[ y \rightarrow \]
Void Functions

\[ x = [40, 20, 10, 30] \]
\[ y = x.sort() \]

\[ x \rightarrow \]
0 ----> 10
1 ----> 20
2 ----> 30
3 ----> 40

\[ y \rightarrow \text{None} \]

Void Functions return None, a special type
Void Functions

x = [40, 20, 10, 30]
y = list(x)
y.sort()

Void Functions return None, a special type
Void Functions

\[ x = [40, 20, 10, 30] \]
\[ y = \text{list}(x) \]
\[ y.\text{sort()} \]

Void Functions return None, a special type
def f(x):
    x = x[1:]
    print x

if __name__ == '__main__':
    u = [1,2,3,4]
    f(u)
    print u

Looks like f deletes the 0-th character in x
def f(x):
    x = x[1:]
    print x

if __name__ == 'blabla'
    u = [1,2,3,4]
    f(u)
    print u

Follow the red dot and watch for impact...
Understanding Function Calls

```python
def f(x):
    x = x[1:]
    print x

if __name__ == 'blabla':
    u = [1, 2, 3, 4]
    f(u)
    print u
```

Parameter `x` initially refers to the same object as `u`
def f(x):
    x = x[1:]
    print x

if __name__ blabla
u = [1,2,3,4]
f(u)
print u

x[1:] creates a new object and x will refer to it
def f(x):
    x = x[1:]
    print x

if __name__ == 'blabla':
    u = [1,2,3,4]
    f(u)
    print u

2 3 4 is printed
Understanding Function Calls

def f(x):
    x = x[1:]
    print x

if __name__ blabla
    u = [1,2,3,4]
    f(u)
    print u

1 2 3 4 is printed
Some Inadvertent Errors

```python
>>> x = [10,20,30]
>>> y = [11,21,31]
>>> z = x+y
>>> print z
[10,20,30,11,21,31]
```
Some Inadvertent Errors

```python
>>> x = [10,20,30]
>>> y = 3*x
>>> Print y
[10,20,30,10,20,30,10,20,3]
```
List Comprehensions

A short cut for setting up “simple” lists

```python
>>> x = [i for i in range(5)]
>>> print x
[0,1,2,3,4]
```
List Comprehensions

A short cut for setting up “simple” lists

```python
>>> x = [1 for i in range(5)]
>>> print x
[1,1,1,1,1,1]
```
List Comprehensions

A short cut for setting up “simple” lists

```python
>>> x = [math.sqrt(i) for i in range(5)]
>>> print x
[0,1,1.414,1.732,2.0]
```
Quickly: Lists of Strings

```python
>>> x = ['Maine','Vermont','New York']
>>> a = x[1]
>>> print a
'Vermont'
>>> c = a[2]
>>> print c
'r'
>>> x[1][2]
'r'
```
Next Problem

Merging Two Sorted Arrays
Into a
Single Sorted Array
Example

\[ x \rightarrow \begin{array}{cccc} 12 & 33 & 35 & 45 \end{array} \]

\[ y \rightarrow \begin{array}{cccc} 15 & 42 & 55 & 65 & 75 \end{array} \]

\[ z \rightarrow \begin{array}{cccccccc} 12 & 15 & 33 & 35 & 42 & 45 & 55 & 65 & 75 \end{array} \]

\( x \) and \( y \) are input

They are sorted

\( z \) is the output
Merging Two Sorted Lists

ix and iy keep track of where we are in x and y
Merging Two Sorted Lists

Do we pick from \( x \)?

\[
x[i_x] \leq y[i_y] \quad ???
\]
Yes. So update ix
**Merge**

$x\rightarrow \begin{bmatrix} 12 & 33 & 35 & 45 \end{bmatrix}$

$y\rightarrow \begin{bmatrix} 15 & 42 & 55 & 65 & 75 \end{bmatrix}$

$z\rightarrow \begin{bmatrix} 12 \end{bmatrix}$

$ix: 1$

$iy: 0$

Do we pick from $x$? $x[ix] \leq y[iy]$ ???
No. So update iy
Do we pick from $x$?  

$x_{ix} \leq y_{iy}$  

???
Yes. So update ix
Do we pick from $x$?

$x[ix] \leq y[iy]$  ???

$ix: 2$

$iy: 1$

$x -> 12 \quad 33 \quad 35 \quad 45$

$y -> 15 \quad 42 \quad 55 \quad 65 \quad 75$

$z -> 12 \quad 15 \quad 33$
Yes. So update ix
Do we pick from \( x \)?

\[
x[i_x] \leq y[i_y] \quad ???
\]
Merge

\[x \rightarrow 12 \ 33 \ 35 \ 45\]

\[y \rightarrow 15 \ 42 \ 55 \ 65 \ 75\]

\[z \rightarrow 12 \ 15 \ 33 \ 35 \ 42\]

No. So update iy...
Merge

Do we pick from $x$? $x[ix] \leq y[iy]$ ???
Yes. So update ix.
Merge

x→ 12 33 35 45

y→ 15 42 55 65 75

z→ 12 15 33 35 42 45

ix: 4
iy: 2

Done with x. Pick from y
Merge

\[ x \rightarrow \begin{array}{cccc} 12 & 33 & 35 & 45 \end{array} \]

\[ y \rightarrow \begin{array}{ccccc} 15 & 42 & 55 & 65 & 75 \end{array} \]

\[ z \rightarrow \begin{array}{cccccc} 12 & 15 & 33 & 35 & 42 & 45 & 55 \end{array} \]

So update iy
Merge

\[ x \rightarrow 12 \quad 33 \quad 35 \quad 45 \]

\[ y \rightarrow 15 \quad 42 \quad 55 \quad 65 \quad 75 \]

\[ z \rightarrow 12 \quad 15 \quad 33 \quad 35 \quad 42 \quad 45 \quad 55 \]

\[ \text{iX:} \quad 4 \]

\[ \text{iY:} \quad 3 \]

Done with \( x \). Pick from \( y \)
Merge

So update iy.
Merge

x→ 12 33 35 45

y→ 15 42 55 65 75

z→ 12 15 33 35 42 45 55 65

Done with x. Pick from y
Merge

\[ x \rightarrow \begin{array}{cccc} 12 & 33 & 35 & 45 \end{array} \]
\[ y \rightarrow \begin{array}{cccc} 15 & 42 & 55 & 65 & 75 \end{array} \]
\[ z \rightarrow \begin{array}{ccccccccc} 12 & 15 & 33 & 35 & 42 & 45 & 55 & 65 & 75 \end{array} \]

Update iy
Merge

\[ x \rightarrow [12, 33, 35, 45] \]
\[ y \rightarrow [15, 42, 55, 65, 75] \]
\[ z \rightarrow [12, 15, 33, 35, 42, 45, 55, 65, 75] \]

ix: 4
iy: 5

All Done
The Python Implementation...
def Merge(x,y):
    n = len(x); m = len(y);
    ix = 0; iy = 0; z = []
    for iz in range(n+m):
        if ix>=n:
            z.append(y[iy]); iy+=1
        elif iy>=m:
            z.append(x[ix]); ix+=1
        elif x[ix] <= y[iy]:
            z.append(x[ix]); ix+=1
        elif x[ix] > y[iy]:
            z.append(y[iy]); iy+=1
    return z

Build z up via repeated appending

x-list exhausted  y-list exhausted  x-value smaller  y-value smaller
def Merge(x, y):
    n = len(x); m = len(y);
    ix = 0; iy = 0; z = []
    for iz in range(n+m):
        if ix >= n:
            z.append(y[iy]); iy+=1
        elif iy >= m:
            z.append(x[ix]); ix+=1
        elif x[ix] <= y[iy]:
            z.append(x[ix]); ix+=1
        elif x[ix] > y[iy]:
            z.append(y[iy]); iy+=1
    return z

len(x)+len(y) is the total length of the merged list

x-list exhausted  y-list exhausted  x-value smaller  y-value smaller
def Merge(x, y):
    u = list(x)  # Make copies of the Incoming lists
    v = list(y)
    z = []
    while len(u) > 0 and len(v) > 0:
        if u[0] <= v[0]:
            g = u.pop(0)
        else:
            g = v.pop(0)
        z.append(g)
    z.extend(u)
    z.extend(v)
    return z
def Merge(x, y):
    u = list(x)
    v = list(y)
    z = []
    while len(u) > 0 and len(v) > 0:
        if u[0] <= v[0]:
            g = u.pop(0)
        else:
            g = v.pop(0)
        z.append(g)
    z.extend(u)
    z.extend(v)
    return z

Implementation Using Pop
Build z up via repeated appending
def Merge(x, y):
    u = list(x)
    v = list(y)
    z = []
    while len(u) > 0 and len(v) > 0:
        if u[0] <= v[0]:
            g = u.pop(0)
        else:
            g = v.pop(0)
        z.append(g)
    z.extend(u)
    z.extend(v)
    return z

Implementation Using Pop

Every “pop” reduces the length by 1. The loop shuts down when one of u or v is exhausted.
Implementation Using Pop

def Merge(x, y):
    u = list(x)  # g gets the popped value
    v = list(y)  # and it is appended to z
    z = []
    while len(u) > 0 and len(v) > 0:
        if u[0] <= v[0]:
            g = u.pop(0)
        else:
            g = v.pop(0)
        z.append(g)
    z.extend(u)
    z.extend(v)
    return z
def Merge(x, y):
    u = list(x)
    v = list(y)
    z = []
    while len(u) > 0 and len(v) > 0:
        if u[0] <= v[0]:
            g = u.pop(0)
        else:
            g = v.pop(0)
        z.append(g)
    z.extend(u)
    z.extend(v)
    return z

Implementation Using Pop
Add what is left in u.
OK if u is the empty list
def Merge(x,y):
    u = list(x)
    v = list(y)
    z = []
    while len(u)>0 and len(v)>0 :
        if u[0] <= v[0]:
            g = u.pop(0)
        else:
            g = v.pop(0)
        z.append(g)
    z.extend(u)
    z.extend(v)
    return z

Add what is left in v. OK if v is the empty list