For Loops and Comprehensions

Aidan Hunt, University of Washington

Learning Objectives

After this lesson, students will be able to

- Interpret for loop syntax in Python
- Understand differences between for loops that use range(), zip(), and enumerate()
- Convert between for loops and list comprehensions

Check-in

Outline

- Check-in (5 mins)
  - Any questions on lists from last time?
- Framing (2 mins)
- Different kinds of for loops (30 minutes)
  - Show students a type of loop and some code, and have them discuss what's going on.
  - Also demo the debugger in Spyder
    - Nested for loop + debugger
    - The enumerate for loop
    - The zip for loop
    - A list comprehension (segue into next section)
- Comprehensions (10 minutes)
  - Show how to translate the comprehension from before into a for loop
  - Translate nested loop
  - Translate other loops
- Summarize (2 mins)
- Chat about upcoming homework (if time)

Framing

Recall from last time:

- Data structures allow us to organize related data in a way that holds meaning.
- Lists are the simplest data structure.
- Use indexing and slicing to get and set values.
- We discussed reference semantics and list methods, too.

Now: our example function.

- We want to perform this operation for each word in our list.
- Right now, it counts the total number of elements in the list, rather than the number of letters in each name.

In [27]:
```python
def countLetters(nameIn):
    '''
    Given an input name (as a string), prints how many letters are in the name.
    '''
    numLetters = len(nameIn)
    print('The name', nameIn, 'has', numLetters, 'letters in it!')

# Define names of people in our class
#name1 = 'Ty'
#name2 = 'Saghar'
#name3 = 'Madeline'
#name4 = 'Ben'
roster = ['Ty', 'Saghar', 'Madeline', 'Ben']

# Call the function for each name
#countLetters(name1)
#countLetters(name2)
#countLetters(name3)
#countLetters(name4)
countLetters(roster)
```

```
The name ['Ty', 'Saghar', 'Madeline', 'Ben'] has 4 letters in it!
```

In [28]:
```python
# Pseudocode:
# countLetters(roster[0])
# countLetters(roster[1])
# countLetters(roster[2])
```

In [29]:
```python
# Call the function for each name
for i in range(len(roster)):
    # print(i) # Uncomment this to see the index increment
    countLetters(roster[i])
```

```
The name Ty has 2 letters in it!
The name Saghar has 6 letters in it!
The name Madeline has 8 letters in it!
The name Ben has 3 letters in it!
```

Key points on syntax

- Use the `for` keyword to declare the loop
- `i` is a variable that changes with each iteration of the loop.
- `range(len(roster))` is where `i` comes from.
    - `len(roster)` is the length of the list `roster`
    - `range(len(roster))` returns a `range` object that generates numbers between 0 and `len(roster)` exclusive

In [30]:
```python
# Example of range on its own
exRange = range(4)
print(exRange)

# Get iterator from range
it = iter(exRange)

# Get values from the iterator
```

```
print(next(it))
print(next(it))
print(next(it))
print(next(it))
```

```
range(0, 4)
0
1
2
3
```

Question: But can we make this even simpler?

In [31]:
```
# Call the function for each name
for name in roster:
    countLetters(name)
```

```
The name Ty has 2 letters in it!
The name Saghar has 6 letters in it!
The name Madeline has 8 letters in it!
The name Ben has 3 letters in it!
```

In this case, we are skipping the index entirely and getting to what we want: the names within the list!

- `name` is the variable whose value changes in each iteration of the loop.
- `roster` is where each `name` comes from

This is pretty awesome, and reads very nicely! This is useful if we don't need the index.

Any questions?

Practice Interpreting For Loops

In Python you'll often see for loops that look a little different, depending on what they are doing.

Discuss with those around you what each of these for loops is doing:

In [32]:
```
# Example 1: nested loop
numList = [1, 2, 3, 4]
list1 = []
for num in numList:
    list2 = []
    for i in range(5):
        list2.append(num * i)

    list1.append(list2)

print(list1)
```

```
[[0, 1, 2, 3, 4], [0, 2, 4, 6, 8], [0, 3, 6, 9, 12], [0, 4, 8, 12, 16]]
```

During each iteration of a for loop, all statements inside the loop are executed. So, if there is a for loop *inside* of another for loop, the inner loop executes completely for each iteration of the outer loop.

Demo debugger and pseudocode for stepping through loop

In [33]:
```
# For loops with enumerate
wordList = ['what', 'is', 'this', 'for', 'loop', 'doing']
for i, word in enumerate(wordList):
    wordList[i] = len(word)

print(wordList)
```

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

The `enumerate` function allows us to simultaneously access both the index of each element and each element itself of a sequence. This can be helpful if:

- you are building a new sequence based on the contents of another sequence (see above)
- you want to update an element of a sequence based on the previous or next element (e.g., an iteration scheme)

In [34]:
```
# Create enumerate object
exEnumerate = enumerate(wordList)

# Convert to list so we can see what it looks like
list(exEnumerate)
```

Out[34]:
```
[(0, 4), (1, 2), (2, 4), (3, 3), (4, 4), (5, 5)]
```

In [58]:
```
# For loops with zip
colorList = ['green', 'blue', 'purple', 'red']
foodList = ['cheese', 'apple', 'sandwich', 'taco']

for color, food in zip(colorList, foodList):
    print("Would you like to eat a", color, food, "?")
```

```
Would you like to eat a green cheese ?
Would you like to eat a blue apple ?
Would you like to eat a purple sandwich ?
Would you like to eat a red taco ?
```

The `zip` function "zips" two sequences together into a list of tuples. The first tuple contains the first elements of each list. The second tuple contains the second elements of each list, and so on.

So, using `zip` is helpful if you want to iterate through two lists simultaneously!

In [59]:
```
# Create zip iterator
exZip = zip(colorList, foodList)

# Convert to list and print to see what the whole thing looks like
list(exZip)
```

Out[59]:
```
[('green', 'cheese'),
 ('blue', 'apple'),
 ('purple', 'sandwich'),
 ('red', 'taco')]
```

In [37]:
```
# For loop....in one line?
numList = [1, 2, 3, 4, 5]
numList = [(num+1)**2 for num in numList]

print(numList)
```

```
[4, 9, 16, 25, 36]
```

This is a special form of loop, called a list comprehension!

List Comprehensions

List comprehensions are compact, readable ways of generating lists without a for loop. They are basically short-hand for for loops, and we can convert back and forth!

In [43]:
```
# Convert the list comprehension to a for loop
numList = [1, 2, 3, 4, 5]
for i, num in enumerate(numList):
    numList[i] = (num+1)**2

print(numList)
```

```
[4, 9, 16, 25, 36]
```

In [62]:
```
# Convert the nested for loop to a list comprehension
numList = [1, 2, 3, 4]
list1 = [[num * i for i in range(5)] for num in numList]

print(list1)
```

```
[[0, 1, 2, 3, 4], [0, 2, 4, 6, 8], [0, 3, 6, 9, 12], [0, 4, 8, 12, 16]]
```

In [44]:
```
# Convert the enumerate loop into a list comprehension
wordList = ['what', 'is', 'this', 'for', 'loop', 'doing']
wordList = [len(word) for word in wordList]

print(wordList)
```

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

In [61]:
```
# Convert the zip loop into a list comprehension
colorList = ['green', 'blue', 'purple', 'red']
foodList = ['cheese', 'apple', 'sandwich', 'taco']
[print("Would you like to eat a", color, food, "?") for color, food in zip(colorList, fo

# Note that a list is automatically created via the list comprehension...
# ...but the print() function doesn't return anything, so its filled with None entries
```

```
Would you like to eat a green cheese ?
Would you like to eat a blue apple ?
Would you like to eat a purple sandwich ?
Would you like to eat a red taco ?
```

Out[61]:
```
[None, None, None, None]
```

When should you use a particular type of for loop/comprehension?

- If you just want to iterate through a list, "standard" for loop (e.g., `for word in wordList`)
- When you need both the index and contents of the list: `enumerate(list1)`
- When you want to iterate through two lists simultaneously: `zip(list1, list2)`
- Use list comprehensions when you want to create a list from another list

Bottom line Use whatever is most readable and works for you to implement.

Wrapping Up

Homework

- Lot's of practice with loops and comprehensions.

Next time