Lists and references

COSC 101, 2018-03-05

Announcements

- Homework #4 due Thursday @ 11pm

Outline

- Warm-up: modifying a list
- List methods
- Iterating over lists
- References and aliasing

Warm-up: Modifying a list

What is the output of each of the following programs?

a) 
   ```python
days = ['Mon', 'Tues', 'Wed', 'Fri']
days[2] = 'Thurs'
days[-1] = 'Sat'
days[0:2] = ['Sun', 'Mon']
print(days)
```

Output:

`['Sun', 'Mon', 'Thurs', 'Sat']`

b) 
   ```python
primes = [2, 3, 5, 7]
primes[2:2] = [11, 13]
primes[0] = [17, 19]
primes[5:] = []
print(primes)
```

Output:

`[[17, 19], 3, 11, 13, 5]`

c) 
   ```python
letters = ['a','e','h','m']
letters[2:2] = ['h']
letters[0:2] = ['q', 'r', 's']
del letters[2:4]
print(letters)
```

Output:

`['q', 'r', 'h', 'm']`

List methods

- Can perform operations on lists, similar to performing operations on strings and turtles
- Common operations
  - `append(value)` --- add `value` to the end of the list
  - `insert(value, index)` --- add `value` at the specified `index`
  - `pop()` --- remove and return the last value in the list
  - `remove(value)` --- remove the first occurrence of `value`
  - `sort()` --- sorts the list
  - `reverse()` --- reverses the order of the list
  - `index(value)` --- returns the index of the first occurrence of `value`
  - `count(value)` --- returns a count of the number of times `value` appears in the list
Rewrite each of the previous programs to use list methods instead of assignment statements.

a) days = ['Mon', 'Tues', 'Wed', 'Fri']
   days.pop(2)
   days.insert(2, 'Thurs')
   days.pop()
   days.append('Sat')
   days.pop(0)
   days.insert(0, 'Sun')
   days.pop(1)
   days.insert(1, 'Mon')
   print(days)

b) primes = [2, 3, 5, 7]
   primes.insert(2, 11)
   primes.insert(3, 13)
   primes.pop(0)
   primes.insert(0, [17, 19])
   primes.pop()
   print(primes)

c) letters = ['a', 'e', 'h', 'm']
   letters.insert(2, 'h')
   letters.pop(0)
   letters.pop(0)
   letters.insert(0, 'q')
   letters.insert(1, 'r')
   letters.insert(2, 's')
   letters.pop(2)
   letters.pop(2)
   print(letters)

Iterating over lists

- Iterating by item --- just like strings
  - Example
    for item in [2, 3, 5]:
      print(item)
  
  Output:
  2
  3
  5

- Iterating by index --- just like strings
  - Example
    primes = [2, 3, 5]
    for index in range(len(primes)):
      print(primes[index])

  Same output as above
What is the output of the following programs?

a)\[
\text{costs} = [1.50, 9.00, 5.00] \\
\text{subtotal} = 0 \\
\text{for index in range(len(costs))}: \\
\text{costs}[\text{index}] = \text{costs}[\text{index}] / 2 \\
\text{subtotal} = \text{subtotal} + \text{costs}[\text{index}] \\
\text{costs}.append(\text{subtotal}*0.1) \\
\text{print(costs)} \\
Output: \\
[0.75, 4.5, 2.5, 0.775]
\]

b)\[
\text{colors} = ['red', 'yellow', 'blue'] \\
\text{combos} = [] \\
\text{for c in range(len(colors)-1)}: \\
\text{mix} = \text{colors}[c] + '+' + \text{colors}[c+1] \\
\text{combos[0:0]} = [\text{mix}] \\
\text{print(combos)} \\
Output: \\
['yellow+blue', 'red+yellow']
\]

References and aliasing

- References
  - Recall: a list is a sequential collection of values
    - Analogy: a folder holds a sequential collection of papers
      - [2, 4, 6]
  - When you assign a list to a variable, the variable becomes a name for the collection
    - Analogy: label folder
      - listA = [2, 4, 6]
  - When you add and remove items, you are changing what is in the collection, but the name still refers to the same collection
    - Analogy: add and remove papers from a folder, but you are not relabeling the folder
      - listA[1] = 40
  - Creating another list creates another sequential collection of values
    - Analogy: get a new folder
      - listB = []
  - Even if you put the same values in each list, the collections are still different
    - Analogy: put the same papers in each folder, but each folder is still separate
      - listB.append(2) \\
      - listB.append(4) \\
      - listB.append(6)

- Aliasing
  - However, if you assign the name of a list to a new name, then you create an alias
    - Analogy: add another label to the list
      - listC = listA
  - Regardless which name you use, you are referring to the same list; hence, changes made using one name will also be reflected if you use the other name to refer to the list
    - Analogy: add and remove papers from the same folder with multiple labels
      - listC[0] = [20] \\
      - listA[2] = [60]
- You can check if two different names refer to the same list using the `is` operator
  - `listA is listC`

- **Cloning**
  - To avoid aliasing, you must create a clone of (part of) the list using the slice operator
    - Analogy: get a new folder with a copy of all of the items in the original folder
    - `listD = listA[:]`

- **Sublists**
  - Recall: can have a list within a list
    - Analogy: put a folder within a folder
    - `listB.append(listA)`
  - If you add one list to another list, then the latter will simply contain a reference to the former

- **Parameters**
  - Recall: at the start of the function, all actual parameters and are assigned to the formal parameters
  - Consequently, list name is assigned another name → formal parameter is an alias
  - Consequence: changing the list using the formal parameter will change the same list as changing the list using the actual parameter