ch_8_assignment

March 28, 2023

Ch_8_assignment

```
[ ]: from IPython.core.interactiveshell import InteractiveShell
     InteractiveShell.ast_node_interactivity = 'all'
```

1 Sotring Collecntions of Data Using Lists

1.1 Storing and Accessing Data in Lists

```
[ ]: # The number of gray whales counted near the Coal Oil Point Natural Reserve in␣
      ↪a two-week period starting on February 24, 2008.
     whales = [5, 4, 7, 3, 2, 3, 2, 6, 4, 2, 1, 7, 1, 3]
     whales
```

```
[ ]: [5, 4, 7, 3, 2, 3, 2, 6, 4, 2, 1, 7, 1, 3]
```

```
[ ]: whales[0]
     whales[1]
     whales[12]
     whales[13]
     third = whales[2]
     print('Third day:', third)
```

```
[ ]: 5

[ ]: 4

[ ]: 1

[ ]: 3

Third day: 7
```

```
[ ]: whales[1001]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[5], line 1
----> 1 whales[1001]

IndexError: list index out of range
```

```
[ ]: whales[-1]
     whales[-2]
     whales[-14]
     third = whales[2]
     print('Thrid day:', third)
```

```
[ ]: 3

[ ]: 1

[ ]: 5

     Thrid day: 7
```

```
[ ]: whales[-15]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[8], line 1
----> 1 whales[-15]

IndexError: list index out of range
```

1.1.1 The Empty List

```
[ ]: # An empty list is a list with no items in it. As with all lists, an empty list␣
      ↪is represented using brackets.
     whales = []
     whales[0]
     whales[-1]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[9], line 3
      1 # An empty list is a list with no items in it. As with all lists, an␣
  ↪empty list is represented using brackets.
      2 whales = []
----> 3 whales[0]
```

```
      4 whales[-1]

IndexError: list index out of range
```

1.1.2 Lists are Heterogeneous

```
[ ]: # Lists can contain any type of data, including integers, strings, and even␣
      ↪other lists.
     krypton = ['Krypton', 'Kr', -157.2, -153.4]
     krypton[1]
     krypton[2]
```

```
[ ]: 'Kr'
```

```
[ ]: -157.2
```

1.2 type Annotations for Lists

```
[ ]: # When writing type contracts for functions, often we'll want to specify that␣
      ↪the values in a list parameter are all of a particular type.
     def average(L: list) -> float:
         """Return the average of the values in L.

         >>> average([1.4, 1.6, 1.8, 2.0])
         1.7
         """
```

```
[ ]: # It would be odd to call it with a list of strings, for example.
     # To address this, Python includes module typing that allows us to specify the␣
      ↪expected type of value contained in a list.
     from typing import List
     def average(L: list[float]) -> float:
         """Return the average of the values in L.

         >>> average([1.4, 1.6, 1.8, 2.0])
         1.7
         """
```

1.3 Modifying Lists

```
[ ]: # That memory model also shows that list objects are mutable. That is, the␣
      ↪contents of a list can be mutated.
     nobles = ['helium', 'none', 'argon', 'krypton', 'xenon', 'radon']
     nobles[1] = 'neon'
     nobles
```

```
[ ]: ['helium', 'neon', 'argon', 'krypton', 'xenon', 'radon']
```

```
[ ]: # In contrast to lists, numbers and strings are immutable.
     name = 'Darwin'
     capitalized = name.upper()
     print(capitalized)
     print(name)
```

```
DARWIN
Darwin
```

1.4 Operations on Lists

```
[ ]: # Some of Python’s built-in functions, such as len, can be applied to lists
     half_lives = [887.7, 24100.0, 6563.0, 14, 373300.0]
     len(half_lives)
     max(half_lives)
     min(half_lives)
     sum(half_lives)
     sorted(half_lives)
     half_lives
```

```
[ ]: 5
```

```
[ ]: 373300.0
```

```
[ ]: 14
```

```
[ ]: 404864.7
```

```
[ ]: [14, 887.7, 6563.0, 24100.0, 373300.0]
```

```
[ ]: [887.7, 24100.0, 6563.0, 14, 373300.0]
```

```
[ ]: # Like strings, lists can be combined using the concatenation (+) operator.
     original = ['H', 'He', 'Li']
     final = original + ['Be']
     final
```

```
[ ]: ['H', 'He', 'Li', 'Be']
```

```
[ ]: # An error occurs when the concatenation operator is applied to a list and a␣
      ↪string
     ['H', 'He', 'Li'] + 'Be'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
```

```
Cell In[24], line 1
----> 1 ['H', 'He', 'Li'] + 'Be'

TypeError: can only concatenate list (not "str") to list
```

[ ]:
```
# You can also multiply a list by an integer to get a new list containing the␣
↪elements from the original list repeated that number of times
metals = ['Fe', 'Ni']
metals * 3
```

[ ]:
```
['Fe', 'Ni', 'Fe', 'Ni', 'Fe', 'Ni']
```

[ ]:
```
# One operator that does modify a list is del, which stands for delete. It can␣
↪be used to remove an item from a list
metals = ['Fe', 'Ni']
del metals[0]
metals
```

[ ]:
```
['Ni']
```

1.4.1 The in Operator in Lists

[ ]:
```
# The in operator can be applied to lists to check whether an object is in a␣
↪list
nobles = ['helium', 'neon', 'argon', 'krypton', 'xenon', 'radon']
gas = input('Enter a gas: ')
print('Enter a gas: ', gas)
if gas in nobles:
    print('{} is noble.'.format(gas))
```

```
Enter a gas:  argon
argon is noble.
```

[ ]:
```
# Unlike with strings, when used with lists, the in operator checks only for a␣
↪single item. This code checks whether the list [1, 2] is an item in the list␣
↪[0, 1, 2, 3]
[1, 2] in [0, 1, 2, 3]
```

[ ]:
```
False
```

1.5 Slicing Lists

[ ]:
```
# list[i:j] is a slice of the original list from index i (inclusive) up to, but␣
↪not including, index j (exclusive).
celegans_phenotypes = ['Emb', 'Him', 'Unc', 'Lon', 'Dpy', 'Sma']
celegans_phenotypes
```

```
useful_markers = celegans_phenotypes[0:4]
useful_markers
```

```
[ ]: ['Emb', 'Him', 'Unc', 'Lon', 'Dpy', 'Sma']
```

```
[ ]: ['Emb', 'Him', 'Unc', 'Lon']
```

```
[ ]: # The first index can be omitted if we want to slice from the beginning of the␣
     ↪list, and the last index can be omitted if we want to slice to the end.
     celegans_phenotypes = ['Emb', 'Him', 'Unc', 'Lon', 'Dpy', 'Sma']
     celegans_phenotypes[:4]
     celegans_phenotypes[4:]
```

```
[ ]: ['Emb', 'Him', 'Unc', 'Lon']
```

```
[ ]: ['Dpy', 'Sma']
```

```
[ ]: # To create a copy of the entire list, omit both indices so that the "slice"␣
     ↪runs from the start of the list to its end.
     celegans_phenotypes = ['Emb', 'Him', 'Unc', 'Lon', 'Dpy', 'Sma']
     celegans_copy = celegans_phenotypes[:]
     celegans_phenotypes[5] = 'Lvl'
     celegans_phenotypes
     celegans_copy
```

```
[ ]: ['Emb', 'Him', 'Unc', 'Lon', 'Dpy', 'Lvl']
```

```
[ ]: ['Emb', 'Him', 'Unc', 'Lon', 'Dpy', 'Sma']
```

1.6 Alias: What's in a Name?

```
[ ]: # An alias is an alternative name for something.
     celegans_phenotypes = ['Emb', 'Him', 'Unc', 'Lon', 'Dpy', 'Sma']
     celegans_alias = celegans_phenotypes
     celegans_phenotypes[5] = 'Lvl'
     celegans_phenotypes
     celegans_alias
```

```
[ ]: ['Emb', 'Him', 'Unc', 'Lon', 'Dpy', 'Lvl']
```

```
[ ]: ['Emb', 'Him', 'Unc', 'Lon', 'Dpy', 'Lvl']
```

```
[ ]: celegans_phenotypes = ['Emb', 'Him', 'Unc', 'Lon', 'Dpy', 'Sma']
     celegans_copy = celegans_phenotypes[:]
     celegans_phenotypes[5] = 'Lvl'
     celegans_phenotypes
     celegans_copy
```

```
[ ]: ['Emb', 'Him', 'Unc', 'Lon', 'Dpy', 'Lv1']
```

```
[ ]: ['Emb', 'Him', 'Unc', 'Lon', 'Dpy', 'Sma']
```

1.6.1 Mutable Parameters

```
[ ]: # Here is a function that takes a list, removes its last item, and returns the␣
      ↪list.
     def remove_last_item(L: list) -> list:
         """Return list L with the last item removed.

         Precondition: len(L) >= 0

         >>> remove_last_item([1, 3, 2, 4])
         [1, 3, 2]
         """
         del L[-1]
         return L

     # In the code that follows, a list is created and stored in a variable; then␣
      ↪that variable is passed as an argument to remove_last_item.
     celegans_markers = ['Emb', 'Him', 'Unc', 'Lon', 'Dpy', 'Lv1']
     remove_last_item(celegans_markers) # When the call on function remove_last_item␣
      ↪is executed, parameter L is assigned the memory address that␣
      ↪celegans_markers contains.
     celegans_markers # That makes celegans_marker and L aliases.
```

```
[ ]: ['Emb', 'Him', 'Unc', 'Lon', 'Dpy']
```

```
[ ]: ['Emb', 'Him', 'Unc', 'Lon', 'Dpy']
```

```
[ ]: from typing import List, Any
     def remove_last_item(L: list[Any]) -> list:
         """Return list L with the last item removed.

         Precondition: len(L) >= 0

         >>> remove_last_item([1, 3, 2, 4])
         [1, 3, 2]
         """
         del L[-1]
         return L
```

1.7 List Methods

```
[ ]: # Lists are objects and thus have methods.
     colors = ['red', 'orange', 'green']
     colors.extend(['black', 'blue'])
     colors

     colors.append('purple')
     colors

     colors.insert(2, 'yellow')
     colors

     colors.remove('black')
     colors
```

```
[ ]: ['red', 'orange', 'green', 'black', 'blue']

[ ]: ['red', 'orange', 'green', 'black', 'blue', 'purple']

[ ]: ['red', 'orange', 'yellow', 'green', 'black', 'blue', 'purple']

[ ]: ['red', 'orange', 'yellow', 'green', 'blue', 'purple']
```

1.8 Working with a List of Lists

```
[ ]: # A list whose items are lists is called a nested list.
     life = [['Canada', 76.5], ['United States', 75.5], ['Mexico', 72.0]]
     life[0]
     life[1]
     life[2]
     print()

     # Since each of these items is also a list, we can index it again, just as we␣
      ↪can chain together method calls or nest function calls
     life[1]
     life[1][0]
     life[1][1]
     print()

     canada = life[0]
     canada
     canada[0]
     canada[1]
     print()
```

```
# As before, any change we make through the sublist reference will be seen when␣
↪we access the main list, and vice versa
canada[1] = 80.0
canada
life
```

```
[ ]: ['Canada', 76.5]

[ ]: ['United States', 75.5]

[ ]: ['Mexico', 72.0]


[ ]: ['United States', 75.5]

[ ]: 'United States'

[ ]: 75.5


[ ]: ['Canada', 76.5]

[ ]: 'Canada'

[ ]: 76.5


[ ]: ['Canada', 80.0]

[ ]: [['Canada', 80.0], ['United States', 75.5], ['Mexico', 72.0]]
```

Reference * Title: Physics Programming Lecture Note (INU) * Author: Jeongwoo Kim, Ph.D. * Availability: https://sites.google.com/view/jeongwookim

---