PALs CSC-152

Section 02, Professor Kim, M-W-F

Tony Stone

November 12, 2025

Lesson VIII

1 More on Collections

1.1 Lists, and Collection Qualities

We have done a good bit of exploring into lists, so we'll treat this section as a bit of a review. This serves as a good start, as a list is about the most straightforward way to understand a collection of items. It's helpful to think about what makes the other collection types different from lists, as opposed to memorizing all of the qualities of them in isolation. When solving a coding problem that requires a collection, you can start by considering a list, and then adapt based on the special qualities of the other collections.

We should also define the terms relevant when considering different collection types. I'll itemize them below, feel free to refer back to this throughout the rest of the lesson.

- **Ordered**—A collection where each element has a defined position.
- **Mutable**—The property of being changeable (Immutability is the property of being unchangeable).
- **Duplicates**—Objects which are identical in memory.

Discussing datatypes can rapidly become a conversation about **data structures**, or the abstract representation of a given datatype. That is a topic you will encounter in a data structures course (duh), or an algorithms course. We will not get bogged down in such topics, we'll simply do our best to understand the collection datatypes as they are implemented in Python.

1.2 Tuples

A tuple is a collection of datatypes much like lists, with the additional quality of being immutable. when you "change" a tuple object, you are not! it is merely a new collection that looks as how you wanted it to be changed (remember the discussion on strings).

Instead of brackets, tuples are defined using parenthesis. Tuples are also ordered, meaning that each element has a position, and therefore we can use our indexing. Here's a very plain example:

```
1 atuple = (1, 2, 3)
2 print(atuple[1])
```

Output:

```
1 2
```

You may be asking: "Why do I care about tuples?" Fair question, and to that I'll turn our attention back to functions, specifically ones where we return several values simultaneously. Say we want to write a function, where given a value, and a boundary size, we return the lower and upper bounds. We can use a comma to pass several values back with the return statement.

```
# Define our bounds function
def bounds(value, boundarySize):
    lowerBound = value - boundarySize
    upperBound = value + boundarySize
    return lowerBound, upperBound

# Find the lower and upper bound centered at 10, of size 2
lb, ub = bounds(10, 2)
print(lb, ub)
```

Output:

```
8, 12
```

Cool Right! Well what is happening behind the scenes, is that we are actually returning a tuple, and unpacking it into the elements in one fell swoop. Don't believe me?

```
# Define our bounds function
def bounds(value, boundarySize):
    lowerBound = value - boundarySize
    upperBound = value + boundarySize
    return lowerBound, upperBound

# Find the lower and upper bound centered at 10, of size 2
result = bounds(10, 2)
print(type(result), result())
```

Output:

```
<class 'tuple'> (8, 12)
```

First, I told you so, second, this is a helpful thing to keep in mind. Tuples are a good way of organizing related things; elements which would be returned in a single function call, e.g. the lower and upper bound from a center.

1.3 Sets

Sets are a personal favorite of mine, as they are really helpful for realizing problems in set theory. In Python, sets are mutable, and can be changed. Instead of using .append() like lists, they use the .add(). Below is a table that consists of the set operations, and then we'll use a whole bunch of them! Sets Thrive on **uniqueness**, meaning they don't contain duplicates of elements. You can add 3 to a set a million times, and there will only ever be one 3 present.

Operation	Symbol/method
Add an item	.add()
Remove an item	.remove()
Union (combine two sets)	|
Difference (elements in one set, and not the other)	−
Symmetric difference (elements in strictly one of two sets)	∧
Intersection (elements in both sets)	&
Set Inclusion	'in'

```
aset = {1, 2, 3}
aset.add(4)
print(aset)
aset.add(5)
print(aset)
aset = aset | {6, 7} # Union
print(aset)
aset = aset | {2, 3} # Union
print(aset)
aset.remove(4)
print(aset)
aset = aset - {1, 2, 3} # Differnce
print(aset)
aset = aset & {7, 8, 9} # Intersection
print(aset)
aset.remove(7)
print(aset)
```

Output:

```
{1, 2, 3, 4}
{1, 2, 3, 4, 5}
{1, 2, 3, 4, 5, 6, 7}
{1, 2, 3, 4, 5, 6, 7}
{1, 2, 3, 5, 6, 7}
{5, 6, 7}
{7}
set()
```

Beware though: Sets are unordered, so the position an element finds itself in may not be what you think it is, and can make indexing a messy task. One of the best applications for sets is for making considerations about a large group of things, and checking what is in those groups. It is a much less order-dependent thing, and by checking that it exists, you can continue! Perhaps a bit of a nothing-example, but we can still visualize the concept. We have two sets, and we union them to get a single instance of all of their shared parts, we check if an element is present and then perform an action based on the result.

```
aset = {1, 2, 3}
bset = {3, 4, 5}

cset = aset | bset
if 4 not in cset:
    cset.add(4)
else:
    cset.add(0)
print(cset)
```

Output:

```
{0, 1, 2, 3, 4, 5}
```

2 Key Terms

- **Ordered**—A collection where each element has a defined position.
- **Mutable**—The property of being changeable (Immutability is the property of being unchangeable).
- **Duplicates**—Objects which are identical in memory.
- **uniqueness**—The quality of being the singular instance of that thing.

3 Availability

- In Lecture:
  - Wednesday: 10:00am - 11:00am
- PAL Sessions:
  - Monday: 2:00pm - 3:00pm
  - Wednesday: 9:00am - 10:00am
  - Wednesday: 2:00pm - 3:00pm

References

[1] Tony Gaddis. *Starting Out with Python*. Pearson, 5th edition, 2021.

[2] GeeksforGeeks. Python Tutorial — Learn Python Programming Language - GeeksforGeeks — geeksforgeeks.org. https://www.geeksforgeeks.org/python/python-programming-language-tutorial/. [Accessed 29-10-2025].

[3] Bradley N. Miller and David L. Ranum. Problem solving with algorithms and data structures using python. https://runestone.academy/ns/books/published/pythonds/index.html, 2014. Interactive online edition — accessed: 2024.

[4] Python Software Foundation. *Python: A dynamic, open source programming language*, 2025. Version 3.x documentation.

[5] W3Schools. W3Schools.com — w3schools.com. https://www.w3schools.com/python/default.asp. [Accessed 29-10-2025].