Python Collection Types

Lists

Lists

- A sequence of items that have unlimited length, known order, can be mixed data types, mutable
    - y = [1,2,3]
    - mylist = [“the”, “cat”, “in”, “the”, “hat”]
    - another_list = [1, “the”, 3.45, True]

List - Defining & Operators

- x = list((1,2,3)) or x = [1,2,3]
  - Creates a list
  - To create a blank list use: x = list() or x = []
- mylist = [4,5,6]
- x + mylist → [1,2,3,4,5,6] (adds both lists together but doesn't assign it to anything)
- x * 3 → [1, 2, 3, 1, 2, 3, 1, 2, 3] (makes 3 copies of list but doesn't assign it to anything)

List - Methods

- x = [1,2,3]
- .append() - adds an item to the end of the list
  - x.append(4) → [1,2,3,4]
- .remove() - removes an item from the list (from the end)
  - x.remove(4) → [1,2,3]
- .pop() - pops an item of the end of the list and returns it
  - x.pop() → 3
  - x.pop(0) → 1 (pop(0) = front of the list)
- .extend() - extends the first list by adding the 2nd list to it
  - y = [9,10]
  - # x = [2]
  - x.extend(y) → [2,9,10]

List - Joining a list together

- **str.join()**
  - Used to concatenate a sequence of strings into one string
  - .join() takes a list as argument
  - separator = “-”
  - sequence = [“join”, “me”, “together”]
  - separator.join(sequence) = “join-me-together”
  - “ ”.join(sequence) = “join me together

Tuple

Tuple

- An immutable ordered list with a known number of elements.
  - Syntax: x = (1,4,6)
  - Immutability refers to the inability to be changed after the original assignment.
  - Tuples, are considered a primitive data type and like all the primitive data types, are immutable.

Set

Set

- An unordered collection of UNIQUE items.
  - Syntax: x = {4,1,6} or x = set((4,1,6))
    - If y = {4,4,6,1} ⟶ y = {4,6,1} (the extra 4 is removed because its not unique item)
  - Cannot update an item only add or remove
    - set.add() ⟶ adds that item to the set
      - x.add(7) -> {1, 4, 6, 7}
    - remove() ⟶ removes that item from the set
      - x.remove(1) -> {6, 4, 7}

Indexing & Slicing

Indexing

- An iterable is any data type that can be used in a sequential fashion to find the next item, which includes string, list, tuple, dictionary, etc.
- We use the iterable property when searching through the various items to find a specific item, which is called indexing:
  - mylist = [“the”, “cat”, “in”, “the”, “hat”]
- Python is ‘zero-based’ so indexing for the first item:
  - mylist[0] → “the”

More Practice with Indexing

- mylist = [“the”, “cat”, “in”, “the”, “hat”]
  - mylist[1] $\longrightarrow$ “cat”
  - mylist[-1] $\longrightarrow$ “hat”
  - mylist[-4] $\longrightarrow$ “cat”
- mystr = ‘python’
  - mystr[0] $\longrightarrow$?
  - mystr[-1] $\longrightarrow$?

Slicing

- To call up a subset/part of a list, we use a slice
- Slice syntax = [# to start with, # to end on (does not include): step]:
  - If either of the first two numbers are left blank - defaults to the start or end of the iterable
  - If the step is left blank - defaults to a step of 1
- Examples: mylist = [“the”, “cat”, “in”, “the”, “hat”]
  - mylist[0:2] ⟶ [“the”, “cat”] (includes items 0 and 1, but not 2)
  - mylist[2:3] ⟶ [“in”] (only include item 2, equivalent to indexing mylist[2])
  - mylist[2:] ⟶ [“in”, “the”, “hat”] (the remainder of the list)
  - mylist[:-1] ⟶ [“the”, “cat”, “in”, “the”] (everything up to the last item)

More Slicing Practice

- Examples: mylist = [“the”, “cat”, “in”, “the”, “hat”]
  - mylist[0:4:2] ⟶ ['the', 'in'] (first item then step of 2)
  - mylist[::-1] ⟶ ['hat', 'the', 'in', 'cat', 'the'] (reverses!)
  - mylist[4:8] ⟶ [‘hat’]
- Example: mystr = ‘Python’
  - mystr[0:2] ⟶ ?
  - mystr[4:6].upper() ⟶ ?
  - mystr[1:5:3] ⟶ ?
  - mystr[::-1] ⟶ ?