BUILT-IN COLLECTIONS: A FIRST LOOK

Python Programming — Auburn University

SEQUENCE TYPES

- Python sequence types: lists, tuples, strings and ranges
  - additional sequence types can be added and some are available in the standard library
- Sequence operations: https://docs.python.org/3/library/stdtypes.html#typesseq-common
- Mutable sequence operations (lists): https://docs.python.org/3/library/stdtypes.html#mutable-sequence-types
- We will cover the sequence types individually…

LISTS

- Array-like structure.
- Indexable (0-based) but with much more flexibility that Java's arrays, for example.
- Expandable. Elements can be inserted into/removed from anywhere in a list and other items will be shifted as necessary.
- Literal lists: `[1, 2, 3]`
- Demo

ELEMENTARY LIST OPERATIONS

- Indexing:

```
lst = [“monkey”, “zebra”, “elephant”]
lst[0]          –> “monkey”
lst[2]          –> “elephant”
```

- Length:

```
len(lst)          –> 3
```

- Min/max (compares with <):

```
min(lst)          –> “elephant”
max(lst)          –> “zebra”
```

- Many others! We’ll cover more later…but see docs if you’re interested.

TUPLES

- Similar to lists but *immutable* (no operation changes the contents of a tuple)
- Literal: `(1,2,"a","monkey")`
- Note tuple with one element requires trailing comma to remove ambiguity:
  `(42,)` is a tuple of length `1` containing the integer 42
  `(42)` is the integer `42`

TUPLE OPERATIONS

- All of the examples from list work **except** assignment to an index:

```
items = (“monkey”, “zebra”, “elephant”)
items[0] = “bear”                  –> ERROR!  tuples are immutable
```

- Demo: Mutable/immutable sequence operations documentation

TUPLE ASSIGNMENT

- "Unpack" tuple, assigning each part to a different variable:
  ```
  a,b,c = (2, 3, 4)
  ```
  assigns `a = 2, b = 3, c = 4`.

- Use * to capture other elements
  ```
  a,b,*rest = (2, 3, 4, 5, 6)
  ```
  assigns `a = 2, b = 3` and `rest = [4, 5, 6]`. Note rest is a **list**.

- Tuple unpacking can be nested:
  ```
  a,b,(c,d) = (2, 3,(4, 5))
  ```

- Many Python functions return tuples. Example:

  - `divmod` returns a tuple with the floored quotient and modulus so:
    ```
    quotient, mod = divmod(11, 3)
    ```
    assigns `quotient = 11 // 3` and `mod = 11 % 3`. (see divmod documentation for details)

TUPLES WITHOUT PARENS

- As long as there is no ambiguity, parentheses can be dropped:

```
a, b = 5+1, 19                # OK
en = enumerate(1,2,3)         # Error, enumerate expects 1 arg
en = enumerate((1,2,3))       # OK
```

- PyCharm will encourage you to remove the parens when they aren't needed.

TUPLES VS LISTS

- Most of the time just use lists :)
- Functions should return tuples (rather than lists) if tuple-assignment seems likely.
- Use a tuple when you want to perform tuple assignment. See assignment lecture:

```
a,b,c = "I", "am", "ironman"
```

- If you want to ensure that the contents of a sequence cannot be changed, use a tuple.
- Tuples can be used as keys in dictionaries and as elements of a set, lists cannot.

SETS

- A set is a collection of items without repetition.
- An item is either in a set or not (can use in operator, just like lists).
- Sets are implemented using hashing (remember data structures!).
- The `in` operator is `O(1)` for sets containing elements with well-behaved hash codes…
- …whereas it is `O(N)` for lists since they use a linear search.
- Set literal: `{“monkey”, “banana”, “tree”}`
- Demo

DICTIONARIES

- Stores a collection of key-value associations.
- Literal syntax: `{“apple”: 5, “banana”: 12, “lemon”: 3}`
- Key must be “hashable”
- Like HashMap in Java
- See https://docs.python.org/3/tutorial/datastructures.html#dictionaries
- Demo