Sequences: strings, tuples, and lists

Strings

A (*character*) *string* `s` is a sequence of characters in single quotes (`'...'`) (or double quotes, `"..."`, which are the same) or in triple double quotes (`"""..."""`), which allow a multi-line string.

- `len(s)` gives its length, e.g. `symbol = 'AMZN'; n = len(symbol); print(f'n={n}.')`
  (Note: "`;`" is a statement separator. I use it for concise notes. It is poor style to use it much.)
- the ith character is `s[i]`, for an `i` in `0` to `n - 1`, e.g. `symbol[0]`, `symbol[n-1]`
- the $(n-i)$th character is `s[-i]`, e.g. `symbol[-1]`
- the *slice* (or *substring*) from `low` to `high - 1` is `s[low:high]` (we can omit `low` or `high`), e.g. `symbol[1:3] # excludes symbol[3]`; also try `symbol[1:]`
- `s + t` joins `s` to string `t`, e.g. `symbol[2] + symbol[1]`
- `s * i` repeats `s` `i` times, e.g. `symbol[0] * 3`
- `s in t` tells whether `s` is in string `t`, e.g. `'MZ' in symbol # also try 'ZM'`
- `s.index(x)` gives the index of the first `x` in `s`, e.g. `symbol.index('Z')`
- `str(object)` creates a string from `object`, e.g. `'n=' + str(n) # try without str() too`
- `help(str)` gives methods we may want later, like `s.capitalize()`, `s.split()`, `s.find()`; e.g. `help(str)`, `symbol.capitalize()`, `'Madison, WI'.split(', ') # try ' ', ',' too`

A string is *immutable*, so `symbol[0] = 'B'` causes an error.

Tuples

A *tuple* is an *immutable* sequence of values, often of varying types. Create a tuple from a comma-separated set of values, usually enclosed in `()`, or via `tuple()`. The string operations, above, work with tuples. e.g.

```
student = ('Badger', 'Bucky', 'junior', 123, ('FIN 310', 'MATH 223', 'CS 410'))
type(student)
type(('apple',)) # tuple of length 1 requires trailing comma
type(('apple'))  # string, not tuple
student[2] = 'senior' # error: tuples are immutable
student = student[0:2] + ('senior',) + student[3:] # change variable, not tuple
```

A function can return only one value, but it can be a tuple. e.g.

```
quotient, remainder = divmod(7, 3) # (an unimportant illustrative function)
print(f'7 divided by 3 yields quotient {quotient} and remainder {remainder}.')
```

Lists

A *list* is a *mutable* sequence of values not necessarily of the same type; typically a list is used for values of the same type. Create a list by enclosing values in `[]`. The string and tuple operations, above, work with lists. e.g.

```
stocks = ['GME', 'AMZN']
list_of_lists = [[0, 0], [1, 1], [2, 5], [3, 9]]
list_of_lists[2][1] = 4; list_of_lists
```

- `.append()` adds its argument as a single value to the end of a list, e.g. `stocks.append('TWTR'); stocks`, `stocks.append(['IBM', 'GOOG']); stocks`
- `.extend()` appends each value of another list, e.g.
  `stocks = ['GME', 'AMZN']; stocks.extend(['IBM', 'GOOG']); stocks`
- `.remove(x)` removes the first occurrence of `x`, e.g. `stocks.remove('IBM'); stocks`
- `.sort()` sorts, e.g. `stocks.sort(); stocks`
- `sorted()` returns a new sorted list, e.g. `stocks = ['GME', 'AMZN']; sorted(stocks); stocks`
- `sum()` adds up the list's values, e.g. `squares = [1, 4, 9]; sum(squares)`
- `.pop(i)` removes and returns the ith value, e.g. `stocks.pop(1); stocks`

Two ways to traverse a sequence with a "`for` *value* `in` *sequence*:" loop

```
sum_squares = 0 # here we use a loop to see what sum() does; run at pythontutor.com
for value in squares: # 1st way: set value to each item in sequence
    sum_squares = sum_squares + value
    print(f'  value={value}, sum_squares={sum_squares}') # indent code 4 spaces

product = 1 # sums start at 0, products at 1
n = len(squares)
for i in range(n): # 2nd way: set i to each index; range(n) is 0, ..., n-1
    product = product * squares[i]
    print(f'  squares[{i}]={squares[i]}, product={product}')
```

e.g. Two more loops: on left, lower-case stock names; on right, find portfolio = $\sum_i \text{price}_i \times \#\text{shares}_i$:

```
lower_stocks = []
for stock in stocks:
    lower_stocks.append(stock.lower())

print(lower_stocks)
```

```
price = (10, 15, 12); n_shares = (1, 2, 5)
portfolio = 0
for i in range(len(price)):
    portfolio += price[i] * n_shares[i]

print(f'portfolio={portfolio}')
```

To learn more, see Think Python's strings, lists, and tuples chapters.