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.

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

A string is immutable, so \( \text{symbol}[0] = 'B' \) 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 \( \text{tuple}() \). The string operations, above, work with tuples. e.g.

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

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

\[
\begin{align*}
\text{quotient}, \text{remainder} &= \text{divmod}(7, 3) \# (an \text{unimportant illustrative function}) \\
\text{print}(f'7 \text{ divided by 3 yields quotient} \{\text{quotient}\} \text{ and remainder} \{\text{remainder}\}').
\end{align*}
\]
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
```

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

```
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.