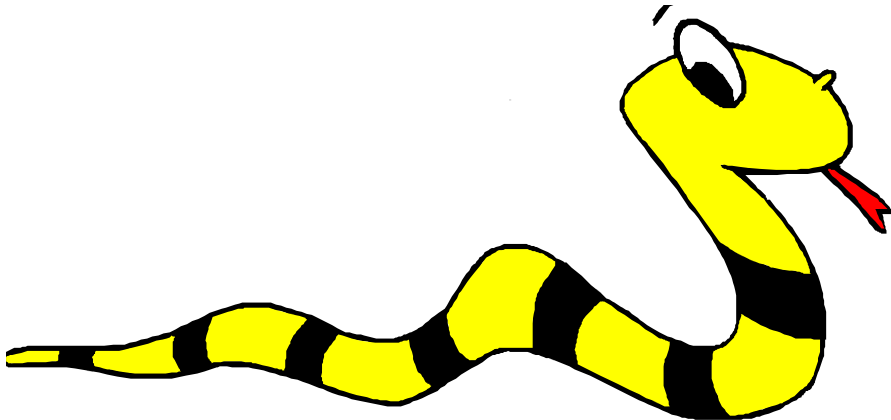


Sequence types: Tuples, Lists, and Strings



Sequence Types

1. Tuple: ('john', 32, [CMSC])

- A simple *immutable* ordered sequence of items
- Items can be of mixed types, including collection types

2. Strings: "John Smith"

- *Immutable*
- Conceptually very much like a tuple

3. List: [1, 2, 'john', ('up', 'down')]

- *Mutable* ordered sequence of items of mixed types

Similar Syntax

- All three sequence types (tuples, strings, and lists) share much of the same syntax and functionality.
- Key difference:
 - Tuples and strings are *immutable*
 - Lists are *mutable*
- The operations shown in this section can be applied to *all* sequence types
 - most examples will just show the operation performed on one

Sequence Types 1

- Define tuples using parentheses and commas

```
>>> tu = (23, 'abc', 4.56, (2,3), 'def')
```

- Define lists are using square brackets and commas

```
>>> li = ["abc", 34, 4.34, 23]
```

- Define strings using quotes (“, ‘, or “””).

```
>>> st = "Hello World"
```

```
>>> st = 'Hello World'
```

```
>>> st = """This is a multi-line  
string that uses triple quotes."""
```

Sequence Types 2

- Access individual members of a tuple, list, or string using square bracket “array” notation
- *Note that all are 0 based...*

```
>>> tu = (23, 'abc', 4.56, (2,3), 'def')
>>> tu[1]      # Second item in the tuple.
'abc'
```

```
>>> li = ["abc", 34, 4.34, 23]
>>> li[1]      # Second item in the list.
34
```

```
>>> st = "Hello World"
>>> st[1]      # Second character in string.
'e'
```

Positive and negative indices

```
>>> t = (23, 'abc', 4.56, (2, 3), 'def')
```

Positive index: count from the left, starting with 0

```
>>> t[1]
```

```
'abc'
```

Negative index: count from right, starting with -1

```
>>> t[-3]
```

```
4.56
```

Slicing: return copy of a subset

```
>>> t = (23, 'abc', 4.56, (2, 3), 'def')
```

Return a copy of the container with a subset of the original members. Start copying at the first index, and stop copying before second.

```
>>> t[1:4]  
( 'abc', 4.56, (2, 3) )
```

Negative indices count from end

```
>>> t[1:-1]  
( 'abc', 4.56, (2, 3) )
```

Slicing: return copy of a =subset

```
>>> t = (23, 'abc', 4.56, (2, 3), 'def')
```

Omit first index to make copy starting from beginning of the container

```
>>> t[:2]  
(23, 'abc')
```

Omit second index to make copy starting at first index and going to end

```
>>> t[2:]  
(4.56, (2, 3), 'def')
```


Copying the Whole Sequence

- `[:]` makes a *copy* of an entire sequence

```
>>> t[ : ]
```

```
(23, 'abc', 4.56, (2,3), 'def')
```

- Note the difference between these two lines for mutable sequences

```
>>> l2 = l1 # Both refer to 1 ref,  
           # changing one affects both
```

```
>>> l2 = l1[ : ] # Independent copies, two  
refs
```

The 'in' Operator

- Boolean test whether a value is inside a container:

```
>>> t = [1, 2, 4, 5]
>>> 3 in t
False
>>> 4 in t
True
>>> 4 not in t
False
```

- For strings, tests for substrings

```
>>> a = 'abcde'
>>> 'c' in a
True
>>> 'cd' in a
True
>>> 'ac' in a
False
```

- Be careful: the *in* keyword is also used in the syntax of *for loops* and *list comprehensions*

The + Operator

The + operator produces a *new* tuple, list, or string whose value is the concatenation of its arguments.

```
>>> (1, 2, 3) + (4, 5, 6)
(1, 2, 3, 4, 5, 6)
```

```
>>> [1, 2, 3] + [4, 5, 6]
[1, 2, 3, 4, 5, 6]
```

```
>>> "Hello" + " " + "World"
'Hello World'
```

The * Operator

- The * operator produces a *new* tuple, list, or string that “repeats” the original content.

```
>>> (1, 2, 3) * 3  
(1, 2, 3, 1, 2, 3, 1, 2, 3)
```

```
>>> [1, 2, 3] * 3  
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
>>> "Hello" * 3  
'HelloHelloHello'
```