CSCE 110: Programming I

David Kebo Hougninou

Basics of Python

Collective Data Structures
lists, sets, dictionaries, tuples
Tuples

A tuple is a sequence of values. The values can be of any type, and they are indexed by integers, so in that respect tuples are similar to lists.

Unlike lists, tuples are immutable.

It is common to enclose tuples in parentheses:

\[ t = ('a', 'b', 'c', 'd', 'e') \]

Creating tuples

Placing all the items inside a parentheses ( ), separated by comma. The parentheses are optional.

A tuple can have items of different datatypes (int, float, list, string, ...)

Syntactically, a tuple is a comma-separated list of values:

\[ t = 'a', 'b', 'c', 'd', 'e' \]

It is common to enclose tuples in parentheses:

\[ t = ('a', 'b', 'c', 'd', 'e') \]
Creating tuples

Note:
A value in parentheses is not a tuple:
\( t_2 = ('a') \) \#t2 is not a tuple, t2 is a string

To create a tuple with a single element, include a final comma:
\( t_1 = ('a',) \) \#t1 is a tuple

Creating tuples: tuple() constructor

The built-in function tuple() creates a tuple.
With no argument, it creates an empty tuple:
\( t = \text{tuple()} \)

If the argument is a sequence (string, list or tuple), the result is a tuple with the elements of the sequence:
\( t = \text{tuple('python')} \)

\[
>>> t \\
('p', 'y', 't', 'h', 'o', 'n')
\]
**Tuples: indexing**

Like strings, each element in a tuple has an index. The index operator `[]` accesses an element in the tuple.

Negative indexing works the same way. Index -1 gets the last item, -2 the second last item etc.

```python
g>>> language = ('p', 'y', 't', 'h', 'o', 'n')
```

```python
... print(language[0])  # first item
... print(language[-6])  # first item
... print(language[5])   # last item
... print(language[-1])  # last item
```

**Tuples: slicing operators**

To slice a range of elements in a tuple, or sub-tuple, use the slicing operators `[:]` or `[:,:]`

```python
seq[:    ]  # [seq[0], seq[1], ..., seq[-1]]
seq[low:]  # [seq[low], seq[low+1], ..., seq[-1]]
seq[:high] # [seq[0], seq[1], ..., seq[high-1]]
seq[low:high] # [seq[low], seq[low+1], ..., seq[high-1]]
seq[:::stride] # [seq[0], seq[stride], ..., seq[-1]]
seq[low::stride] # [seq[low], seq[low+stride], ..., seq[-1]]
seq[:high::stride] # [seq[0], seq[stride], ..., seq[high-1]]
seq[low:high::stride] # [seq[low], seq[low+stride], ..., seq[high-1]]
```
Tuples: slicing operators

To slice a range of elements in a tuple, or sub-tuple, use the slicing operators [:] or [:]

With a negative stride, the ordering changes since we are counting down.

```python
seq[::-stride]  # [seq[-1], seq[-1-stride], ..., seq[0]]
seq[high::-stride]  # [seq[high], seq[high-stride], ..., seq[0]]
seq[low::-stride]  # [seq[-1], seq[-1-stride], ..., seq[low+1]]
seq[high:low:-stride]  # [seq[high], seq[high-stride], ..., seq[low+1]]
```

Tuples: slicing operators

To slice a range of elements in a tuple, or sub-tuple, use the slicing operators [:] or [:] with a positive or negative stride.

```python
>>> language = ('p', 'y', 't', 'h', 'o', 'n')
>>> language[3:6:1]
('h', 'o', 'n')
>>> language[2:0:-1]
('t', 'y')
>>> language[::-1]
('n', 'o', 'h', 't', 'y', 'p')
```
Tuples: changing a tuple

Unlike lists, tuples are immutable.
Elements of a tuple cannot be changed once we create the tuple object.
We can assign a tuple to different values (reassignment).

```python
>>> roster = (4, 2, 3, [6, 5])
... # roster[1] = 9  # invalid!
... roster[3][0] = 9
... print(roster)
... roster = ('p', 'y', 't', 'h', 'o', 'n')
... print(roster)
... (4, 2, 3, [9, 5])
('p', 'y', 't', 'h', 'o', 'n')
```

Tuples: tuple assignment

How can we swap two variables \(a\) and \(b\)?
How can we swap two variables \(a\) and \(b\)?

With conventional assignments, use a **temporary** variable.

\[
\begin{align*}
\text{temp} &= a \\
a &= b \\
b &= \text{temp}
\end{align*}
\]

Are there other methods?

A **tuple assignment** is more elegant:

\[
a, b = b, a
\]

**Note**: # of variables on the left must be equal to # of values on the right:

e.g. \(a, b = b, a, c\) is incorrect
Tuples: concatenation and repetition

The \texttt{+} operator concatenate two tuples.

The \texttt{*} operator repeats a tuple \texttt{n} times.

```
>>> print((1, 2, 3) + (4, 5, 6))
... print((1, 2, 3) * 3)
... print(("dice", 6) * 4)
...
(1, 2, 3, 4, 5, 6)
(1, 2, 3, 1, 2, 3, 1, 2, 3)
('dice', 6, 'dice', 6, 'dice', 6, 'dice', 6)
```

Tuple: delete \texttt{del}

Because a tuple is immutable, we cannot change or delete the items in a tuple.

The \texttt{del} keyword deletes the entire tuple.

```
>>> roster = ('p', 'y', 't', 'h', 'o', 'n')
... # del roster[3] #invalid
... del roster
... print(roster)
...
Traceback (most recent call last):
  File "", line 1, in <module>
NameError: name 'roster' is not defined
```
Tuples: count() and index()

tuple.count(x) counts the number of times the item x appears.  
tuple.index(x) returns the 1st index where the item x is found.

```python
>>> roster = ('p', 'y', 't', 'h', 'o', 'n', 'l', 'o', 'g')
... print(roster.count('p'))
... print(roster.index('o'))
1
4
```
# Tuples: built-in functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all()</td>
<td>Return True if all elements of the tuple are true (or if the tuple is empty).</td>
</tr>
<tr>
<td>any()</td>
<td>Return True if any element of the tuple is true. If the tuple is empty, return False.</td>
</tr>
<tr>
<td>enumerate()</td>
<td>Return an enumerate object. It contains the index and value of all the items of tuple as pairs.</td>
</tr>
<tr>
<td>len()</td>
<td>Return the length (the number of items) in the tuple.</td>
</tr>
<tr>
<td>tuple()</td>
<td>Convert an iterable (list, string, set, dictionary) to a tuple.</td>
</tr>
<tr>
<td>max()</td>
<td>Return the largest item in the tuple.</td>
</tr>
<tr>
<td>min()</td>
<td>Return the smallest item in the tuple.</td>
</tr>
<tr>
<td>sorted()</td>
<td>Return a new sorted tuple (does not sort the tuple itself).</td>
</tr>
<tr>
<td>sum()</td>
<td>Return the sum of all elements in the tuple.</td>
</tr>
</tbody>
</table>