06.2_Lists_Tuples

September 27, 2023

1 Introduction to Python for Open Source Geocomputation

- Instructor: Dr. Wei Kang

Content:

- Lists - continued
- Tuples

2 Standard Data Types in Python - lists

Category of Data type	Data type	Example
Numeric, scalar	Integer	1
	Floats	1.2
	Complex	1.5+0.5j
	Booleans	True
Container	strings	“Hello World”
	List	[1, “Hello World”]
	Tuple	(1, “Hello World”)
	Set	{1, “Hello World”}
	Dictionary	{1: “Hello World”, 2: 100}

2.0.1 Iteration with `for` statements on a list

A Python `for` statement iterates over the items of a sequence. Say you have a list called `fruits` containing a sequence of strings with fruit names:

`fruits = ['apple', 'banana', 'orange', 'cherry', 'mandarin']`

you can write a statement like

`for fruit in fruits:`

to do something with each item in the list.

[1]:
```
fruits = ['apple', 'banana', 'orange', 'cherry', 'mandarin']
for fruit in fruits:
    print(fruit)
```

```
apple
banana
orange
cherry
mandarin
```

[2]:
```
fruits = ['apple', 'banana', 'orange', 'cherry', 'mandarin']
for i in fruits:
    print(i)
```

```
apple
banana
orange
cherry
mandarin
```

[3]:
```
fruits = ['apple', 'banana', 'orange', 'cherry', 'mandarin']
for fruit in fruits:
    print(fruits)
```

```
['apple', 'banana', 'orange', 'cherry', 'mandarin']
['apple', 'banana', 'orange', 'cherry', 'mandarin']
['apple', 'banana', 'orange', 'cherry', 'mandarin']
['apple', 'banana', 'orange', 'cherry', 'mandarin']
['apple', 'banana', 'orange', 'cherry', 'mandarin']
```

[4]:
```
fruits = ['apple', 'banana', 'orange', 'cherry', 'mandarin']

for fruit in fruits:
    print("Eat your", fruit, "everday")
    #print function can accept many values separated by comma (spaces are␣
 ↪inserted between values)
```

```
Eat your apple everday
Eat your banana everday
Eat your orange everday
Eat your cherry everday
Eat your mandarin everday
```

We can use `for` loop to update each element according to some rule in a list.

In the following example, we try to multiply each element in a numerical list by 2:

[5]:
```
list_numbers = [1,2,3,4]
list_numbers
```

[5]:
```
[1, 2, 3, 4]
```

[6]:
```
list_numbers * 2
```

[6]:
```
[1, 2, 3, 4, 1, 2, 3, 4]
```

Using `*` directly between the list and number `2` does not work as this will do the repetition of the list.

We can `for` loop to update each element according to some rule (multiple it by `2`) in a list:

[7]:
```
list_numbers = [1,2,3,4]
for i in range(len(list_numbers)):
    list_numbers[i] = list_numbers[i] * 2 #assign the calculated number to each␣
 ↪item of the list
    print(i, list_numbers)
list_numbers
```

```
0 [2, 2, 3, 4]
1 [2, 4, 3, 4]
2 [2, 4, 6, 4]
3 [2, 4, 6, 8]
```

[7]:
```
[2, 4, 6, 8]
```

2.0.2 *Translate that!*

- What is a list in python? What are its properties?

2.1 List Methods

- `list.append()`: adds a new element to the end of a list
- `list.extend()`: takes a list as an argument and appends all of the elements
- `list.sort()`: arranges the elements of the list from low to high (ascending order)
- `list.reverse()`: reverse the list
- `list.remove()`: remove the given element in a list
- `list.pop()`: Remove and return item at index (default last).

Most list methods are **void**:

- "return" `NoneType`

Most list methods are **in-place** methods:

- modify the original list object

```
[8]: y=[]
```

```
[9]: y
```

```
[9]: []
```

```
[10]: y.append(10)
```

```
[11]: y
```

```
[11]: [10]
```

```
[12]: z = y.append(10)
```

```
[13]: type(z)
```

```
[13]: NoneType
```

```
[14]: y
```

```
[14]: [10, 10]
```

```
[15]: x=[1,2,3]
```

```
[16]: x.append(y)
```

```
[17]: x
```

```
[17]: [1, 2, 3, [10, 10]]
```

```
[18]: y
```

```
[18]: [10, 10]
```

```
[19]: x.extend(y)
```

```
[20]: x
```

```
[20]: [1, 2, 3, [10, 10], 10, 10]
```

```
[21]: x.extend(10)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[21], line 1
----> 1 x.extend(10)
```

```
TypeError: 'int' object is not iterable
```

[22]:
```
x.extend([10])
```

[23]:
```
x
```

[23]:
```
[1, 2, 3, [10, 10], 10, 10, 10]
```

[24]:
```
x.append(10)
```

[25]:
```
x
```

[25]:
```
[1, 2, 3, [10, 10], 10, 10, 10, 10]
```

[26]:
```
x.append([10])
```

[27]:
```
x
```

[27]:
```
[1, 2, 3, [10, 10], 10, 10, 10, 10, [10]]
```

[28]:
```
y
```

[28]:
```
[10, 10]
```

[29]:
```
x + y
```

[29]:
```
[1, 2, 3, [10, 10], 10, 10, 10, 10, [10], 10, 10]
```

Note the subtle difference between the two methods. Sometimes you will want to use append, and other times extend is what you need.

[30]:
```
x=[ 7, 1, 3,12]
```

[31]:
```
x.sort()
```

[32]:
```
x
```

[32]:
```
[1, 3, 7, 12]
```

[33]:
```
x.reverse()
```

[34]:
```
x
```

[34]:
```
[12, 7, 3, 1]
```

[35]:
```
x.remove(7)
x
```

[35]:
```
[12, 3, 1]
```

[36]:
```
x.remove(7)
x
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[36], line 1
----> 1 x.remove(7)
      2 x

ValueError: list.remove(x): x not in list
```

2.1.1 *Translate that!*

- What is an in-place method in python?

2.1.2 Lists as Stacks with `list.pop()`

A stack is a linear data structure that follows the principle of **Last In First Out (LIFO)**.

[37]:
```
x = [1,2,3]
```

[38]:
```
x.pop()
```

[38]:
```
3
```

[39]:
```
x
```

[39]:
```
[1, 2]
```

[40]:
```
x.pop()
```

[40]:
```
2
```

[41]:
```
x
```

[41]:
```
[1]
```

2.1.3 Lists as Queues with `list.pop(0)`

A queue is a linear data structure that follows the principle of First In First Out (FIFO) order

[42]:
```
x = [1,2,3]
```

[43]:
```
x.pop(0)
```

[43]:
```
1
```

[44]:
```
x
```

[44]:
```
[2, 3]
```

[45]:
```
x.pop(0)
```

[45]:
```
2
```

[46]:
```
x
```

[46]:
```
[3]
```

[47]:
```
x.pop(0)
```

[47]:
```
3
```

[48]:
```
x
```

[48]:
```
[]
```

2.2 Lists and Strings

- spliting a string into a list of substrings `str.split([delimiter])`
- combine a list of substrings into a string `delimiter.join(list)`

[49]:
```
s = 'spam-spam-spam'
```

[50]:
```
s.split("-")
```

[50]:
```
['spam', 'spam', 'spam']
```

[51]:
```
list_s = s.split("-")
list_s
```

[51]:
```
['spam', 'spam', 'spam']
```

[52]:
```
"-".join(list_s)
```

[52]:
```
'spam-spam-spam'
```

[53]:
```
" ".join(list_s)
```

[53]:
```
'spam spam spam'
```

[54]:
```
"--".join(list_s)
```

[54]:
```
'spam--spam--spam'
```

[55]:
```
";;;;;".join(list_s)
```

[55]:
```
'spam;;;;;spam;;;;;spam'
```

2.3 Further readings

- tutorial on Lists

3 Standard Data Types in Python - Tuples

Category of Data type	Data type	Example
Numeric, scalar	Integer	1
	Floats	1.2
	Complex	1.5+0.5j
	Booleans	True
Container	strings	"Hello World"
	List	[1, "Hello World"]
	Tuple	(1, "Hello World")
	Set	{1, "Hello World"}
	Dictionary	{1: "Hello World", 2: 100}

3.1 Tuples in python

- Similar to lists
  - Ordered sequence
  - each item/element can be of any type
- Exception: immutable

3.2 Creating Tuples

- assignment statement: `t = (1,2,"a")`
  - Having the comma(s) is very important
- function `tuple()`

[56]:
```
t = (1,2,3,'a','b','stella')
```

[57]:
```
type(t)
```

[57]:
```
tuple
```

[58]:
```
t
```

[58]:
```
(1, 2, 3, 'a', 'b', 'stella')
```

[59]:
```
s=1,2,3,'a','b','stella'
s
```

[59]:
```
(1, 2, 3, 'a', 'b', 'stella')
```

[60]:
```
type(s)
```

[60]:
```
tuple
```

[61]:
```
x=1,
x
```

[61]:
```
(1,)
```

[62]:
```
type(x)
```

[62]:
```
tuple
```

[63]:
```
y = 1
```

[64]:
```
type(y)
```

[64]:
```
int
```

[65]:
```
t2 = ('a')
t2
```

[65]:
```
'a'
```

[66]:
```
type(t2)
```

[66]:
```
str
```

[67]:
```
t3 = ('a', )
t3
```

[67]:
```
('a',)
```

[68]:
```
type(t3)
```

[68]:
```
tuple
```

Creating a tuple using the built-in function `tuple()`. The input needs to be iterable (container data type).

[69]:
```
tuple()
```

[69]:
```
()
```

[70]:
```
tuple("python")
```

[70]:
```
('p', 'y', 't', 'h', 'o', 'n')
```

[71]:
```
tuple([1,2,3])
```

[71]:
```
(1, 2, 3)
```

3.2.1 indexing and slicing tuples

- similar to lists and strings

[72]:
```
s=1,2,3,'a','b','stella'
s
```

[72]:
```
(1, 2, 3, 'a', 'b', 'stella')
```

[73]:
```
s[0]
```

[73]:
```
1
```

[74]:
```
s[1:-1]
```

[74]:
```
(2, 3, 'a', 'b')
```

3.2.2 Two Tuple Methods

- `tuple.count()`: Return number of occurrences of value.
- `tuple.index()`: Return first index of value.

[75]:
```
mytupe = 'a', 'b', 'c'
mytupe
```

[75]:
```
('a', 'b', 'c')
```

[76]:
```
mytupe.count('a')
```

[76]:
```
1
```

[77]:
```
mytupe.index('b')
```

[77]:
```
1
```

[78]:
```
mytupe.index('f')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[78], line 1
----> 1 mytupe.index('f')

ValueError: tuple.index(x): x not in tuple
```

3.3 Tuple Operations

- Concatenation with + (similar to strings)
- Repetition with * (similar to strings)

[79]:
```
mytupe = 'a', 'b', 'c'
histupe = 1,2,3
```

[80]:
```
mytupe + histupe
```

[80]:
```
('a', 'b', 'c', 1, 2, 3)
```

[81]:
```
histupe * 4
```

[81]:
```
(1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3)
```

[82]:
```
mytupe * 4
```

[82]:
```
('a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c')
```

3.3.1 Tuples are immutable!

[83]:
```
mytupe
```

[83]:
```
('a', 'b', 'c')
```

[84]:
```
mytupe[0] = "c"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[84], line 1
----> 1 mytupe[0] = "c"

TypeError: 'tuple' object does not support item assignment
```

3.3.2 Tuples are Nestable

- similar to lists

- Note that while the tuple is immutable, if it contains any elements that are mutable (e.g., lists) we can change the elements of the mutable elements of the tuple.

[85]:
```
b=(1,2,3)
t=(b,'a','melissa')
t
```

[85]:
```
((1, 2, 3), 'a', 'melissa')
```

[86]:
```
t[0][0]
```

[86]:
```
1
```

[87]:
```
t[0][0]=100
t
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[87], line 1
----> 1 t[0][0]=100
      2 t

TypeError: 'tuple' object does not support item assignment
```

[88]:
```
l=[1,2,3]
t=(l,'a','melissa')
t
```

[88]:
```
([1, 2, 3], 'a', 'melissa')
```

[89]:
```
t[0][0]=100
t
```

[89]:
```
([100, 2, 3], 'a', 'melissa')
```

[90]:
```
l=(1,2,3)
t=(l,'a','melissa')
t
```

[90]:
```
((1, 2, 3), 'a', 'melissa')
```

[91]:
```
t[0]='d'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[91], line 1
----> 1 t[0]='d'
```

```
TypeError: 'tuple' object does not support item assignment
```

[92]:
```
t[1]='d'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[92], line 1
----> 1 t[1]='d'

TypeError: 'tuple' object does not support item assignment
```

3.3.3 Converting Between Lists and Tuples

- convertinng from list to tuple: `tuple()`
- convertinng from tuple to list: `list()`

[93]:
```
list_a = [1,2,3]
```

[94]:
```
tuple(list_a)
```

[94]:
```
(1, 2, 3)
```

[95]:
```
list(tuple(list_a))
```

[95]:
```
[1, 2, 3]
```

3.3.4 Group exercise

Write code to change the value of the first item in the tuple `mytupe = ('a', 'b', 'c')` to `'d'`

```
mytupe = ('a', 'b', 'c')
```

When you are done, raise your hand!

[96]:
```
mytupe[0] = "d"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[96], line 1
----> 1 mytupe[0] = "d"

TypeError: 'tuple' object does not support item assignment
```

[97]:
```
mytupe = ('a', 'b', 'c')
```

[98]:
```
mytupe_list = list(mytupe)
mytupe_list
```

[98]:
```
['a', 'b', 'c']
```

[99]:
```
mytupe_list[0] = "d"
```

[100]:
```
mytupe_list
```

[100]:
```
['d', 'b', 'c']
```

[101]:
```
mytupe = tuple(mytupe_list)
mytupe
```

[101]:
```
('d', 'b', 'c')
```

[102]:
```
("c", ) + mytupe[1:]
```

[102]:
```
('c', 'b', 'c')
```

[103]:
```
list_mytupe = list(mytupe)
list_mytupe[0] = "d"
list_mytupe
```

[103]:
```
['d', 'b', 'c']
```

[104]:
```
mytupe = tuple(list_mytupe)
```

[105]:
```
mytupe
```

[105]:
```
('d', 'b', 'c')
```

3.3.5 *Translate that!*

- What is a tuple in python?
- What are the differences between a list and a tuple in python?

3.4 Further readings

- Lists and Tuples in Python

[ ]:

[ ]: