Python
Tuples

Learning Objectives

- Introduce and discuss tuples
- Identify the similarities and differences between tuples and lists

Lists and Sequences

- **Recall:** lists are a sequence of values connected by a common name

```
grades = [76, 65, 98]
```

- Lists have many methods to manipulate their contents

```
grades.append(83)
grades.sort()
print(grades)
```

```
[65, 76, 83, 98]
```

- **Methods** are functions attached to some object and are accessed via the . operator

Tuples

- Tuples are sequences of values much like lists with one very key and very important difference:
  - Lists are mutable, they can be changed
  - Tuples are immutable, they cannot be changed
    - Though they may store mutable items
  - Tuples are written as sequences of values separated by commas, sometimes in parenthesis

List vs Tuples

Lists

| **append**(...) | L.append(object) -> None -- append object to end |

| **clear**(...) | L.clear() -> None -- remove all items from L |

| **copy**(...) | L.copy() -> list -- a shallow copy of L |

| **count**(...) | L.count(value) -> integer -- return number of occurrences of value |

| **extend**(...) | L.extend(iterable) -> None -- extend list by appending elements from the iterable |

| **index**(...) | L.index(value, [start, [stop]]) -> integer -- return first index of value. | Raises ValueError if the value is not present. |

| **insert**(...) | L.insert(index, object) -- insert object before index | | pop(...) | L.pop([index]) -> item -- remove and return item at index (default last). | Raises IndexError if list is empty or index is out of range. |

| **remove**(...) | L.remove(value) -> None -- remove first occurrence of value. | Raises ValueError if the value is not present. |

| **reverse**(...) | L.reverse() -- reverse *IN PLACE* |

| **sort**(...) | L.sort(key=None, reverse=False) -> None -- stable sort *IN PLACE*

Tuples

| **count**(...) | T.count(value) -> integer -- return number of occurrences of value |

| **index**(...) | T.index(value, [start, [stop]]) -> integer -- return first index of value. | Raises ValueError if the value is not present.

Declaring a Tuple

- Declaring a tuple can be accomplished by invoking the tuple() function
- Notice the difference in the brackets
  - Square brackets [] indicate a list
  - Curly brackets {} indicate a dictionary
  - Parenthesis () indicate a tuple

```
my_list = list()
my_dictionary = dict()
my_tuple = tuple()
print(my_list)
print(my_dictionary)
print(my_tuple)
```

```
[]
{}
()
```

Declaring a Tuple

- Like with lists and dictionaries, a parenthesis alone can create a tuple
- Since it cannot be changed, an empty tuple is not terribly useful
- Tuples can also be created by a value and a comma as my_tuple=4,

```
my_list = []
my_dictionary = {}
my_tuple = ()
print(my_list)
print(my_dictionary)
print(my_tuple)
```

```
[]
{}
()
```

Using Tuples

- Most list operations can be used such as slices and the len function work on tuples as they do on lists
- Attempting to change a value in a tuple generates an error

```
my_tuple = (1, 3, 4, 5, 6 )
print(my_tuple)
print(my_tuple[1:4])
my_tuple[2] = 5
```

```
(1, 3, 4, 5, 6)
(3, 4, 5)

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-9-3a5deeab04ab> in <module>()
      2 print(my_tuple)
      3 print(my_tuple[1:4])
----> 4 my_tuple[2] = 5

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

Using Tuples

- It is possible to copy and concatenate tuples
  - Combining these operations, allow the simulation of changing a member
  - This is expensive, changing a list accesses one value. Copying a tuple changes the number of values which exist in the tuple.

```
my_tuple = (1, 3, 4, 5, 6 )
my_tuple = my_tuple[:2] + (5,) + my_tuple[3:]
print(my_tuple)
```

```
(1, 3, 5, 5, 6)
```

Tuple Assignments

- It is possible use a Tuple on the left-hand side of an assignment operator
- The left and right sides must be balanced, each tuple must have the same number of members

```
a, b = 4, 5
print(a)
print(b)
a, b = b, a
print(a)
print(b)
a, b = 2, 4, 6
```

```
4
5
5
4
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-13-e54fffeb952d> in <module>()
      5 print(a)
      6 print(b)
----> 7 a, b = 2, 4, 6

ValueError: too many values to unpack (expected 2)
```

Tuple Assignments

- Functions and methods which return a tuple can be used with Tuple assignment
  - Ex: split returns a list of the values in a string separated by some string

```
value = '3pi/4'
(numerator, denominator) = value.split('pi/')
print(int(numerator)/int(denominator))

line = "john,smith,003034,A"
(fname,lname,sid,grade) = line.split(',')
print(fname)
print(lname)
print(sid)
print(grade)
```

```
0.75
john
smith
003034
A
```

Tuple Assignments

- Though a function can only return one value, that value can be a tuple
- Combined with a matching Tuple Assignment, the appearance of returning multiple values is possible
- Note: if the values passed in are not changed, functions can frequently work on lists and tuples interchangeably

```
def minmax(seq):
    return (min(seq),max(seq))

values = [1, 3, 5, 6, 8, 978]
names = "Bob", "Jim", "April", "Jill"
minv, maxv = minmax(values)
print(minv)
print(maxv)
print("-----")

minv, maxv = minmax(names)
print(minv)
print(maxv)
```

```
1
978
-----
April
Jim
```

Variable Arguments

- By prefacing an argument to a function with an * asterisk, all values are gathered into a tuple
- The number of arguments in the function call can be arbitrary

```
def minmax(*seq):
    return (min(seq),max(seq))

minv, maxv = minmax(4,5,65,979,31,8,64)
print(minv)
print(maxv)
```

```
4
979
```

Splitting Tuples

- Corresponding to the gather operation, a tuple can be scattered to its individual values in a function call by using the same * asterisk operator
  - Note the first call works, the vals tuple separates into two discrete values and then are passed into the function
  - The second call doesn't perform the scatter and fails

```
def product(a, b):
    return a * b

vals = 13, 54
print("1:" + str(product(*vals)))
print("2:" + str(product(vals)))
```

```
1:702

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-30-6647386632c4> in <module>()
      4 vals = 13, 54
      5 print("1:" + str(product(*vals)))
----> 6 print("2:" + str(product(vals)))

TypeError: product() missing 1 required positional argument: 'b'
```

Lists and Tuples

- Two sequences can be joined, element by element using the zip function
- zip returns a sequence of pairs comprised of one element from each sequence in a zip object
- The zip object is an iterator. Iterators are used to iterate over a set of data
- Note: the sequence of pairs the length of the shorter seq

```
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9]
names = ('Jesus','Jose','Martin')
for pair in zip(nums, names):
    print(pair)
```

```
(1, 'Jesus')
(2, 'Jose')
(3, 'Martin')
```

Only prints three entries since names only has three entries

Dictionaries and Tuples

- There is a method for dictionary objects, items, which returns a sequence of (key, value) tuples
- The result is a dict_items iterator which can be used to iterate over the dictionary
- A list of tuples can be used to initialize a dictionary

```
grades = {'Jesus':'A','Jose':'B','Martin':'C'}
print(grades)
print(grades.items())
for key, value in grades.items():
    print(key + "->" + value)
```

```
{'Jesus': 'A', 'Jose': 'B', 'Martin': 'C'}
dict_items([('Jesus', 'A'), ('Jose', 'B'), ('Martin', 'C')])
Jesus->A
Jose->B
Martin->C
```

```
vals = [('Jesus', 'A'), ('Jose', 'B'), ('Martin', 'C')]
grades = dict(vals)
print(grades)
```

```
{'Jesus': 'A', 'Jose': 'B', 'Martin': 'C'}
```

Dictionaries and Tuples

- Recall: the key to a dictionary can be any immutable value
- Recall: tuples are immutable

```
grades = {('Jesus', 'Smith'):'A',('Jose', 'Garcia'):'B',('Martin','Francis'):'C'}
print(grades)
print(grades['Jesus','Smith'])
print("")
print("{0:10s}{1:10s}{2:5s}".format("First", "Last", "Grade"))
print("------------------------")
for first, last in grades:
    print("{0:10s}{1:10s}{2:^5s}".format(first, last, grades[first, last]))
```

```
{('Jesus', 'Smith'): 'A', ('Jose', 'Garcia'): 'B', ('Martin', 'Francis'): 'C'}
A

First     Last      Grade
------------------------
Jesus     Smith       A
Jose      Garcia      B
Martin    Francis     C
```

Resources

- Bryan Burlingame's notes
- Downey, A. (2016) *Think Python, Second Edition* Sebastopol, CA: O'Reilly Media
- (n.d.). 3.7.0 Documentation. *5. Data Structures — Python 3.7.0 documentation.* Retrieved October 30, 2018, from https://docs.python.org/3/tutorial/datastructures.html