

CMSC201

Computer Science I for Majors

Lecture 16 – Tuples

Prof. Katherine Gibson

Prof. Jeremy Dixon

Last Class We Covered

- “Good” code
- Code design
 - Readability
 - Adaptability
- Top-down design
- Modular development

Any Questions from Last Time?

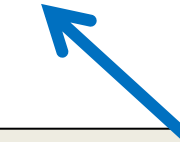
Today's Objectives

- Learn about the *tuple* data structure in Python
- Perform basic operations with tuples including:
 - Creation
 - Conversion
 - Repetition
 - Slicing
 - Traversing
- Use tuples in functions (as return values)

Tuples

The *Tuple* Data Structure

- In Python, a tuple is an *immutable* sequence of values



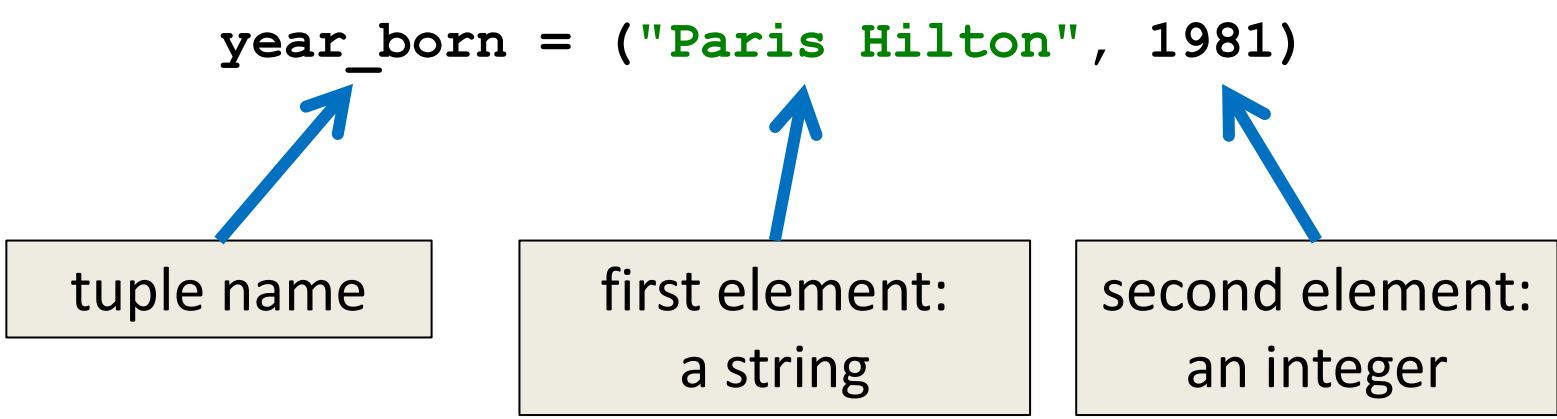
What does
immutable
mean?

Tuples are immutable which means
you *cannot update or change* the
values of tuple elements

The *Tuple* Data Structure

- Each value in the tuple is an *element* or *item*
- Elements can be any Python data type
 - Tuples can mix data types
 - Elements can be nested tuples

```
year_born = ("Paris Hilton", 1981)
```



tuple name

first element:
a string

second element:
an integer

Creating Tuples

Creating Tuples

- The empty tuple is written as two parentheses containing nothing

```
tup1 = ()
```

- To cast a list as a tuple, you use `tuple()`

```
myList = [5, 15, 23]
```

```
myTuple = tuple(myList)
```

```
print(type(myTuple))
```

```
<class 'tuple'>
```

Creating Tuples

```
numbers = (1, 2, 3, 4)
```

```
print (numbers)
```

```
(1, 2, 3, 4)
```

```
cheeses = ('swiss', 'cheddar',  
           'ricotta', 'gouda')
```

```
print (cheeses)
```

```
('swiss', 'cheddar', 'ricotta', 'gouda')
```

Creating Tuples

```
t1 = ('a')          a <class 'str'>
print (t1, type(t1))
```

Is this a tuple?

```
t2 = ('a',)       ('a',) <class 'tuple'>
print (t2, type(t2))
```

Tuples with one element require a comma

Creating Tuples

```
t3 = tuple('a')
```

```
print (t3, type(t3)) ('a',) <class 'tuple'>
```

```
empty = tuple()      ()
```

```
print (empty)
```

Creating Tuples

```
aList = [1, 2, 3, 4]
aTuple = tuple(aList)
print (aTuple)
```

What does this output?

(1, 2, 3, 4)

```
aStr = 'parrot'      ('p', 'a', 'r', 'r', 'o', 't')
aTuple2 = tuple(aStr)
print (aTuple2)
```

Indexing and Slicing Tuples

Tuple Indexing

- Just like other sequences (strings), elements within a tuple are indexed

```
cheeses = ('swiss', 'cheddar',  
          'ricotta', 'gouda')  
print (cheeses[0])
```

What does this do?

```
cheeses[0] = 'swiss'
```

Nothing! (an error)

Tuples are immutable.

Slicing a Tuple

- Like other sequences, tuples can be sliced
- Slicing a tuple creates a new tuple. It does not change the original tuple.

```
cheeses = ('swiss', 'cheddar',  
           'ricotta', 'gouda')
```

```
print (cheeses[1:4])
```

What does this
output?

```
('cheddar', 'ricotta', 'gouda')
```


Tuple Operations

Operations on Tuples

- Tuples support all the standard sequence operations, including:
 - Membership tests (using the **in** keyword)
 - Comparison (element-wise)
 - Iteration (e.g., in a **for** loop)
 - Concatenation and repetition
 - The **len()** function
 - The **min()** and **max()** functions

Membership Tests (**in**)

- In Python, the *in* keyword is used to test if a sequence (list, tuple, string etc.) contains a value.
 - Returns *True* or *False*

```
a = [1, 2, 3, 4, 5]
print(5 in a)
print(10 in a)
```

True

False

What does this
output?

Comparison

- In Python 3.3, we can use the comparison operator, `==`, to do tuple comparison
 - Returns *True* or *False*

```
tuple1, tuple2 = (123, 'xyz'), (456, 'abc')
tuple3 = (456, 'abc')
print (tuple1==tuple2)
print (tuple2==tuple3)
```

What does this
output?

False
True

Iteration

```
teams = ((1, 'Ravens'), (2, 'Panthers'),  
         (5, 'Eagles'), (7, 'Steelers'))
```

Notice tuple of tuples

```
for (index, name) in teams:  
    print(index, name)
```

```
1 Ravens  
2 Panthers  
5 Eagles  
7 Steelers
```

What does this
output?

Iteration

```
t = [('a', 0), ('b', 1), ('c', 2)]  
for letter, number in t:  
    print (number, letter)
```

Notice list of tuples

```
0 a  
1 b  
2 c
```

What does this
output?

Concatenation (+)

- The + operator returns a new tuple that is a concatenation of two tuples

```
a = (1, 2, 3)
```

```
b = (4, 5, 6)
```

```
c = a + b
```

```
print (a, b, c)
```

What does this output?

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

Repetition (*)

- The `*` operator returns a new tuple that repeats the tuple.

```
a = (1, 2, 3)
b = (4, 5, 6)
print (a*2, b)
```

What does this
output?

(1, 2, 3, 1, 2, 3) (4, 5, 6)

len () Functions

- The method `len ()` returns the number of elements in the tuple.

```
tuple0 = ()
```

```
print(len(tuple0))
```

```
tupleA = ("UMBC", "is", "the", "best")
```

```
print(len(tupleA))
```

What does this
output?

0

4

`min()` and `max()` Functions

- `max(tuple)`
 - Returns item from the tuple with max value
- `min(tuple)`
 - Returns item from the tuple with min value

```
myTuple = tuple('parrot')  
print (myTuple)
```

```
print (min (myTuple) )  
print (max (myTuple) )
```

```
('p', 'a', 'r', 'r', 'o', 't')  
a  
t
```

What does this
output?

Tuples and Functions (`return`)

Tuples and functions

- Python functions (as is true of most languages) can only return one value
 - But... but... we've returned multiple values before!
- If multiple objects are packaged together into a tuple, then the function can return the objects as a single tuple
- Many Python functions return tuples

Example: min_max.py

```
# Returns the smallest and largest  
# elements of a sequence as a tuple  
def min_max(t):  
    return min(t), max(t)
```

```
seq = [12, 98, 23, 74, 3, 54]  
print (min_max(seq))
```

What does this
output?

```
string = 'She turned me into a newt!'  
print (min_max(string))  
myMin, myMax = min_max(string)
```

(3, 98)
(' ', 'w')

Passing Tuples as Parameters

Passing Tuples as Parameters

- A parameter name that begins with `*` gathers all the arguments into a tuple
- This allows functions to take a variable number of parameters
 - So we can call the function with one, or two, or twenty parameters!
- (An actual parameter is also called an argument)

Example

```
def printall(*args):  
    print (args)
```

Actual Parameters
(or Arguments)



```
printall(1, 2.0, 'three')
```


Example: pointless.py

```
def pointless(required, *args):  
    print ('Required:', required)  
    if args:  
        print('Others: ', args)
```

What does this
output?

```
pointless(1)
```

```
pointless(1, 2)
```

```
pointless(1, 2.0, 'three')
```

```
pointless(1, 2.0, 'three', [4])
```

```
Required: 1
```

```
Required: 1
```

```
Others: (2,)
```

```
Required: 1
```

```
Others: (2.0, 'three')
```

```
Required: 1
```

```
Others: (2.0, 'three', [4])
```

Any Other Questions?

Announcements

- Homework 7 is due
 - Due by Monday (April 4th) at 8:59:59 PM
- Project 1 will come out Monday night

- Next time: Dictionaries