CS 134:
Tuples and Sorting

Announcements & Logistics

- **HW 5** due today at 11pm
- **Lab 4** part 1 feedback returned on Friday
  - Try to fix any issues before submitting Part 2
- **Lab 4 Part 2** today/tomorrow
  - Due Wed/Thur at 11 pm
- **Midterm reminder**:
  - Thur Mar 17: Slots: 6 - 7:30 pm, 8 - 9:30 pm in NSB B11/002
  - Two rooms reserved (one for reduced distractions/extra time)
- **Midterm review**: Tue Mar 15: 7 - 8:30 pm in TPL 203
  - Midterm practice problems will be released soon

Do You Have Any Questions?

Looking Ahead

- **No HW posted this week**
  - We'll post practice midterm questions instead
- **Lab next week**
  - Short lab on debugging strategies
  - Start and finish during lab!
  - No need to start in advance
- **Things to review in preparation for the midterm**
  - Review lab solutions and HW questions
  - Review Jupyter notebooks and slides
  - Discuss practice midterm questions
- **No class on Fri Mar 18**

Last Time

- Learned about **aliasing** in Python
  - Need to be careful with aliasing when using lists due to mutability
- Discussed ways to create "new" lists (true copies):

```
newList = myList[:] # slicing

newList = [el for el in myList] # list comprehension
```

- Discussed while loops
  - Needed for ranked-choice voting on Lab 4 Part 2

Today's Plan

- Today we will discuss a new ***immutable*** sequence: **tuples**
- Revisit sorting and default sorting behavior
- Discuss how we can override the default sorting behavior

Tuples: An Immutable Sequence

- Tuples are an **immutable sequence of values** (almost like immutable lists) separated by commas and enclosed within parentheses **( )**

In [1]:
```
# string tuple
names = ('Shikha', 'Jeannie', 'Kelly', 'Lida')

# num tuple
primes = (2, 3, 5, 7, 11)

# singleton
num = (5,)

# parens are optional
values = 5, 6

# empty tuple
emp = ()
```

A tuple of size 1 is called a singleton. Note the syntax.

Tuples as Immutable Sequences

- Tuples, like strings, support any sequence operation that ***does not involve mutation***: e.g,
  - `len()` function: returns number of elements in tuple
  - `[]` indexing: access specific element
  - `+, *`: tuple concatenation
  - `[:]`: slicing to return subset of tuple (as a new tuple)
  - `in` and `not in`: check membership
  - `for loop`: iterate over elements in tuple

Multiple Assignment and Unpacking

- Tuples support simple and nifty syntax for assigning multiple values at once, and also for "unpacking" sequence values

```
>>> a, b = 4, 7
# reverse the order of values in tuple
>>> b, a = a, b
>>> harryInfo = ['Harry Potter', 11, True]
# tuple assignment to “unpack” list elements
>>> name, age, glasses = harryInfo
```

- Note that the preceding line is a more concise (preferred) way of writing:

```
>>> name = harryInfo[0]
>>> age = harryInfo[1]
>>> glasses = harryInfo[2]
```

Multiple Return from Functions

- Tuples also come in handy as well when returning multiple values from functions

In [1]:
```
# multiple return values as a tuple
def arithmetic(num1, num2):
    '''Takes two numbers and returns the sum and product'''
    return num1 + num2, num1 * num2
```

In [2]:
```
arithmetic(10, 2)
```

Out[2]:
```
(12, 20)
```

In [3]:
```
type(arithmetic(3, 4))
```

Out[3]:
```
tuple
```

Conversion between Sequences

- The functions **`tuple()`**, **`list()`**, and **`str()`** let us convert between sequences

```
In [4]: word = "Williamstown"

In [5]: charList = list(word)

In [6]: charList
Out[6]: ['W', 'i', 'l', 'l', 'i', 'a', 'm', 's', 't', 'o', 'w', 'n']

In [7]: charTuple = tuple(charList)

In [8]: charTuple
Out[8]: ('W', 'i', 'l', 'l', 'i', 'a', 'm', 's', 't', 'o', 'w', 'n')

In [9]: list((1, 2, 3, 4, 5)) # tuple to list
Out[9]: [1, 2, 3, 4, 5]
```

Conversion between Sequences

- The functions **`tuple()`**, **`list()`**, and **`str()`** let us convert between sequences

```
In [40]: numRange = range(len(word))

In [41]: list(numRange)
Out[41]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

In [42]: str(list(numRange))
Out[42]: '[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]'

In [43]: str(('hello', 'world'))
Out[43]: "('hello', 'world')"
```

- See Jupyter for more examples

Sorting Tuples and More

sorted( )

- Python has a built-in function for sorting sequences: `sorted()`
- `sorted()` is a function (not a method!!) that takes a sequence (string, list, tuple) and returns a ***new sorted sequence as a list***
- By default, `sorted()` sorts the sequence in **ascending order** (for numbers) and alphabetical (dictionary) order for strings
- `sorted()` ***does not alter the sequence*** it is called on and it always returns the type `list`

```
In [1]: nums = (42, -20, 13, 10, 0, 11, 18)
        sorted(nums)

Out[1]: [-20, 0, 10, 11, 13, 18, 42]

In [2]: letters = ('a', 'c', 'e', 'p', 'z')
        sorted(letters)

Out[2]: ['a', 'c', 'e', 'p', 'z']
```

sorted( )

- **`sorted(string)`** returns a sorted **`list`** of **`strings`** (not string!)

```
In [1]: # sorted() will sort the characters in the string and return a list
        sorted("Rohit")

Out[1]: ['R', 'h', 'i', 'o', 't']

In [2]: sorted("Jeannie")

Out[2]: ['J', 'a', 'e', 'e', 'i', 'n', 'n']

In [4]: sorted("*hello! world!*")

Out[4]: [' ', '!', '!', '*', '*', 'd', 'e', 'h', 'l', 'l', 'l', 'o', 'o', 'r', 'w']
```

Sorting Strings

- Strings are sorted based on the **ASCII values** of their characters
- ASCII stands for *"American Standard Code for Information Interchange"*
- Common character encoding scheme for electronic communication (that is, anything sent on the Internet!)
- Special characters come first, followed by capital letters, then lowercase letters
- Characters encoded using integers from `0–127`
- Can use Python functions `ord()` and `chr()` to work with these:
  - `ord(str):` takes a character and returns its ASCII value as `int`
  - `chr(int):` takes an ASCII value as `int` and returns its corresponding character (`str`)

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	*[NULL]*	32	20	*[SPACE]*	64	40	@	96	60	`
1	1	*[START OF HEADING]*	33	21	!	65	41	A	97	61	a
2	2	*[START OF TEXT]*	34	22	"	66	42	B	98	62	b
3	3	*[END OF TEXT]*	35	23	#	67	43	C	99	63	c
4	4	*[END OF TRANSMISSION]*	36	24	$	68	44	D	100	64	d
5	5	*[ENQUIRY]*	37	25	%	69	45	E	101	65	e
6	6	*[ACKNOWLEDGE]*	38	26	&	70	46	F	102	66	f
7	7	*[BELL]*	39	27	'	71	47	G	103	67	g
8	8	*[BACKSPACE]*	40	28	(	72	48	H	104	68	h
9	9	*[HORIZONTAL TAB]*	41	29	)	73	49	I	105	69	i
10	A	*[LINE FEED]*	42	2A	*	74	4A	J	106	6A	j
11	B	*[VERTICAL TAB]*	43	2B	+	75	4B	K	107	6B	k
12	C	*[FORM FEED]*	44	2C	,	76	4C	L	108	6C	l
13	D	*[CARRIAGE RETURN]*	45	2D	-	77	4D	M	109	6D	m
14	E	*[SHIFT OUT]*	46	2E	.	78	4E	N	110	6E	n
15	F	*[SHIFT IN]*	47	2F	/	79	4F	O	111	6F	o
16	10	*[DATA LINK ESCAPE]*	48	30	0	80	50	P	112	70	p
17	11	*[DEVICE CONTROL 1]*	49	31	1	81	51	Q	113	71	q
18	12	*[DEVICE CONTROL 2]*	50	32	2	82	52	R	114	72	r
19	13	*[DEVICE CONTROL 3]*	51	33	3	83	53	S	115	73	s
20	14	*[DEVICE CONTROL 4]*	52	34	4	84	54	T	116	74	t
21	15	*[NEGATIVE ACKNOWLEDGE]*	53	35	5	85	55	U	117	75	u
22	16	*[SYNCHRONOUS IDLE]*	54	36	6	86	56	V	118	76	v
23	17	*[ENG OF TRANS. BLOCK]*	55	37	7	87	57	W	119	77	w
24	18	*[CANCEL]*	56	38	8	88	58	X	120	78	x
25	19	*[END OF MEDIUM]*	57	39	9	89	59	Y	121	79	y
26	1A	*[SUBSTITUTE]*	58	3A	:	90	5A	Z	122	7A	z
27	1B	*[ESCAPE]*	59	3B	;	91	5B	[	123	7B	{
28	1C	*[FILE SEPARATOR]*	60	3C	<	92	5C	\	124	7C	|
29	1D	*[GROUP SEPARATOR]*	61	3D	=	93	5D	]	125	7D	}
30	1E	*[RECORD SEPARATOR]*	62	3E	>	94	5E	^	126	7E	~
31	1F	*[UNIT SEPARATOR]*	63	3F	?	95	5F	_	127	7F	*[DEL]*

An aside: sort() vs sorted()

- `.sort()` ***method*** is **only for lists** and sorts by mutating the list in place; invoked using dot notation
- `sorted()` ***function*** can be used to **sort any sequence** (strings, lists, tuples). It always **returns a new sorted list**, and does NOT modify the original sequence

Example:

```
list1 = [6, 3, 4];  list2 = [6, 3, 4]
list1.sort() # sort list1 by mutating values
sorted(list2) # returns a *new* sorted list
```

list1 Before	list1 After	list2 Before	list2 After
[6, 3, 4]	[3, 4, 6]	[6, 3, 4]	[6, 3, 4]

Does not change!

Sorting Tuples and Lists

- Sorting a list of (or a tuple of) tuples with `sorted()` sorts elements in ascending order by their first item
- If there is a tie, Python breaks the tie by comparing the second items
- If the second items are also tied, it compares the third items, and so on

```
In [1]: fruits = [(12, 'apples'), (5, 'kiwis'), (4, 'bananas'), (27, 'grapes')]
        sorted(fruits)

Out[1]: [(4, 'bananas'), (5, 'kiwis'), (12, 'apples'), (27, 'grapes')]
```

```
In [2]: pairs = [(4, 5), (0, 2), (12, 1), (11, 3)]
        sorted(pairs)

Out[2]: [(0, 2), (4, 5), (11, 3), (12, 1)]
```

- Note: The same is true for lists and lists of lists
- This sorting behavior is referred to as **lexicographical sorting**

Sorting Tuples and Lists

- Sorting a list of (or a tuple of) tuples with `sorted()` sorts elements in ascending order by their first item
- If there is a tie, Python breaks the tie by comparing the second items
- If the second items are also tied, it compares the third items, and so on

```
In [3]: triples = [(1, 2, 3), (1, 3, 2), (2, 2, 1), (1, 2, 1)]
        sorted(triples)

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

```
In [4]: characters = [(8, 'a', '$'), (7, 'c', '@'),
                      (7, 'b', '+'), (8, 'a', '!')]

        sorted(characters)

Out[4]: [(7, 'b', '+'), (7, 'c', '@'), (8, 'a', '!'), (8, 'a', '$')]
```

Question: How do we sort based on the second/third item in tuples? Or sort in reverse order?

Changing the Default Sorting Behavior

- To understand the `sorted()` function more, lets read its documentation

```
In [5]: help(sorted)
```

```
Help on built-in function sorted in module builtins:

sorted(iterable, /, *, key=None, reverse=False)
    Return a new list containing all items from the iterable in ascending order.

    A custom key function can be supplied to customize the sort order, and the
    reverse flag can be set to request the result in descending order.
```

- First parameter is an ***iterable***, meaning, any object over which we can iterate (list, string, tuple, range).
- `sorted()` takes a optional parameter **key** which specifies a function, that for each element, determines how it should be compared to other elements
- `sorted()` takes an optional boolean parameter **reverse** (which by default is set to **False**)

Reverse Sorting

- Let's consider the optional `reverse` parameter to `sorted()`
- We can sort sequences in reverse order by setting this parameter to be True

```
In [2]: sorted([8, 2, 3, 1, 3, 1, 2], reverse=True)
Out[2]: [8, 3, 3, 2, 2, 1, 1]

In [3]: sorted(['a', 'c', 'e', 'p', 'z'], reverse=True)
Out[3]: ['z', 'p', 'e', 'c', 'a']

In [4]: fruits = [(12, 'apples'), (5, 'kiwis'), (4, 'bananas'), (27, 'grapes')]
        sorted(fruits, reverse=True)
Out[4]: [(27, 'grapes'), (12, 'apples'), (5, 'kiwis'), (4, 'bananas')]
```

Sorting with a `key` function

- Now suppose we have a list of tuples that we want to sort by something *other* than the first item
- Example: We have a list of course tuples, where the first item is the course name, second item is the enrollment cap, and third item is the term (Fall/Spring).

```
courses = [('CS134', 74, 'Spring'), ('CS136', 60, 'Spring'),
           ('AFR206', 30, 'Spring'), ('ECON233', 30, 'Fall'),
           ('MUS112', 10, 'Fall'), ('STAT200', 50, 'Spring'),
           ('PSYC201', 50, 'Fall'), ('MATH110', 74, 'Spring')]
```

- Suppose we want to sort these courses by their **capacity** (second element)
- We can accomplish this by supplying the `sorted()` function with a `key` function that tells it how to compare the tuples to each other

Sorting with a `key` function

- **Defining a key function explicitly:**
  - We can define an explicit `key` function that, when given a tuple, returns the parameter we want to sort the tuples with respect to

```
def capacity(courseTuple):
    '''Takes a sequence and returns item at index 1'''
    return courseTuple[1]
```

  - Once we have defined this function, we can pass it as a `key` when calling `sorted()`

```
# can tell sorted to sort by capacity instead
sorted(courses, key=capacity)
```

Sorting with a **key** function

- **Defining a key function explicitly:**
  - We can define an explicit **key** function that, when given a tuple, returns the parameter we want to sort the tuples with respect to

```
def capacity(courseTuple):
    '''Takes a sequence and returns item at index 1'''
    return courseTuple[1]
```

```
# we can tell sorted() to sort by capacity instead
sorted(courses, key=capacity)
```

```
[('MUS112', 10, 'Fall'),
 ('AFR206', 30, 'Spring'),
 ('ECON233', 30, 'Fall'),
 ('STAT200', 50, 'Spring'),
 ('PSYC201', 50, 'Fall'),
 ('CS136', 60, 'Spring'),
 ('CS134', 74, 'Spring'),
 ('MATH110', 74, 'Spring')]
```