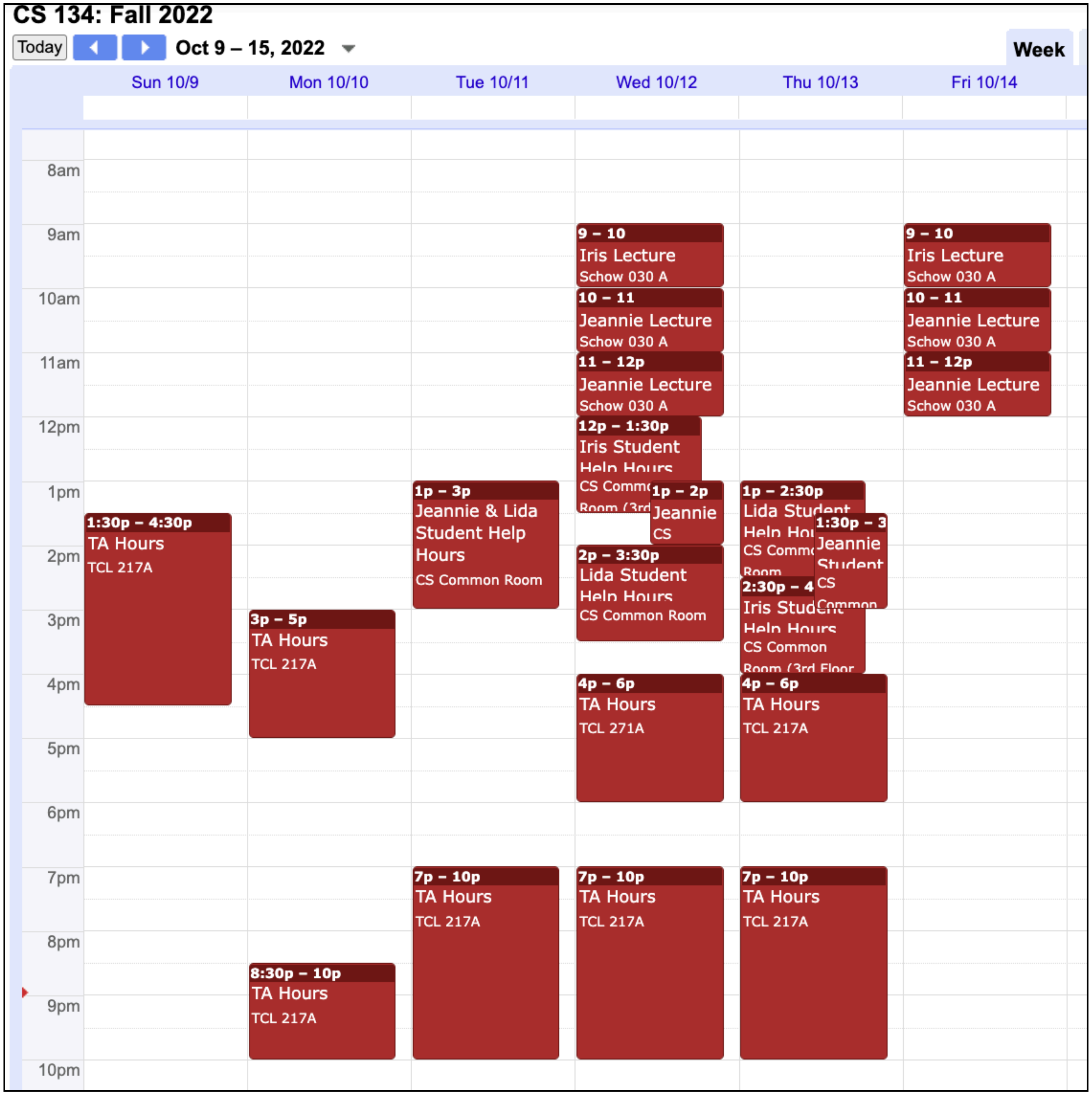
CS134: Tuples & Sorting

Announcements & Logistics

- **HW 5** due Monday at 10pm - last HW before midterm
- **Lab 4**
  - **Part 1**: Feedback returned this afternoon
  - **Part 2**: Due next Wed/Thur at 10 pm
- **Midterm reminder**: Thur Oct 20: 6-7:30 pm and 8-9:30 pm
- **Midterm review**: Tue Oct 18: 8-9:30 pm
  - Midterm practice problems will be released soon
- **Student help hours** for next week are now posted on webpage
  - Lots of hours on Tue, Wed, and Thur!
  - If the CS common room gets too crowded, we'll move to TCL 217A/216

Student Help Hours Next Week


CS 134: Fall 2022

Oct 9 – 15, 2022

Sun 10/9
- 1:30p – 4:30p TA Hours TCL 217A

Mon 10/10
- 3p – 5p TA Hours TCL 217A
- 8:30p – 10p TA Hours TCL 217A

Tue 10/11
- 1p – 3p Jeannie & Lida Student Help Hours CS Common Room
- 7p – 10p TA Hours TCL 217A

Wed 10/12
- 9 – 10 Iris Lecture Schow 030 A
- 10 – 11 Jeannie Lecture Schow 030 A
- 11 – 12p Jeannie Lecture Schow 030 A
- 12p – 1:30p Iris Student Help Hours CS Comm Room (3rd
- 1p – 2p Jeannie CS
- 2p – 3:30p Lida Student Help Hours CS Common Room
- 4p – 6p TA Hours TCL 271A
- 7p – 10p TA Hours TCL 217A

Thu 10/13
- 1p – 2:30p Lida Student Help Hou CS Comm Room
- 1:30p – 3 Jeannie Student CS Common
- 2:30p – 4 Iris Student Help Hours CS Common Room (3rd Floor
- 4p – 6p TA Hours TCL 217A
- 7p – 10p TA Hours TCL 217A

Fri 10/14
- 9 – 10 Iris Lecture Schow 030 A
- 10 – 11 Jeannie Lecture Schow 030 A
- 11 – 12p Jeannie Lecture Schow 030 A

Looking Ahead

- **No HW posted next week**
  - We'll post practice midterm questions instead
- **Lab on Oct 17/18**
  - Short lab on debugging strategies
  - Start and finish during scheduled lab session!
  - 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 Oct 21** (regardless of Mountain Day)

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

Recap: Loops

1. Initialize a variable used in the test condition
2. Keyword that indicates the beginning of the loop
3. Test condition that causes the loop to end when False
4. Colon that indicates the end of the loop definition
5. Within the loop body (indented!), update the variable used in the test condition

Initialize a variable for test condition

```python
def printHalves(n):
    while n > 0:
        print(n)
        n = n//2
```

Colon

Test condition causing loop to end

Update test condition variable in loop body

Keyword for beginning of loop

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 **( )**

```
# string tuple
>>> names = ("Jeannie", "Iris", "Lida")

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

# singleton
>>> num = (5, )

# parentheses are optional
>>> values = 5, 6

# empty tuple
>>> emp = ()
```

A tuple of size 1 is called a singleton. Note the (funky) 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 a simple 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
# tuple assignment to “unpack” list elements
>>> cbInfo = ['Charlie Brown', 8, False]
>>> name, age, glasses = cbInfo
```

- Note that the preceding line is just a more concise way of writing:

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

Multiple Return from Functions

- Tuples come in handy when returning multiple values from functions

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

```
>>> arithmetic(10, 2)
(12, 20)
>>> type(arithmetic(3, 4))
<class 'tuple'>
```

Conversion between Sequences

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

```
>>> word = "Williamstown"
>>> charList = list(word) # string to list
>>> charList
['W', 'i', 'l', 'l', 'i', 'a', 'm', 's', 't', 'o', 'w', 'n']
>>> charTuple = tuple(charList) # list to tuple
>>> charTuple
('W', 'i', 'l', 'l', 'i', 'a', 'm', 's', 't', 'o', 'w', 'n')
>>> list((1, 2, 3, 4, 5)) # tuple to list
[1, 2, 3, 4, 5]
```

Conversion between Sequences

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

```
>>> str(('hello', ‘world')) # tuple to string
"('hello', ‘world')"
>>> numRange = range(12)
>>> list(numRange) # range to list
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>>> str(list(numRange)) # range to list to string
'[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]'
```

- See Jupyter for more examples

Sorting Tuples & More

sorted( )

- `sorted()` is a built-in Python 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 always returns the type `list`

```
>>> nums = (42, -20, 13, 10, 0, 11, 18) # tuple of ints
>>> sorted(nums) # this returns a list!
[-20, 0, 10, 11, 13, 18, 42]
>>> letters = ('a', 'c', 'z', 'b', 'Z', ‘A')
>>> sorted(letters)
['A', 'Z', 'a', 'b', 'c', 'z']
```

sorted( )

- `sorted(string)` returns a sorted `list` of `strings` (or more specifically, characters). It does not return a string!

```
>>> sorted("Iris")
['I', 'i', 'r', 's']
>>> sorted("Jeannie")
['J', 'a', 'e', 'e', 'i', 'n', 'n']
>>> sorted("*hello!*")
['!', '*', '*', 'e', 'h', 'l', 'l', 'o']
```

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`)

DO NOT MEMORIZE!

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

```
>>> fruits = [(12, ‘apples'), (4, 'bananas'), (27, 'grapes')]
>>> sorted(fruits)
[(4, ‘bananas'), (12, 'apples'), (27, 'grapes')]
>>> pairs = [(4, 5), (0, 2), (12, 1), (11, 3)]
>>> sorted(pairs)
[(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

```
>>> triples = [(1, 2, 3), (2, 2, 1), (1, 2, 1)]
>>> sorted(triples)
[(1, 2, 1), (1, 2, 3), (2, 2, 1)]
>>> chars = [(8, 'a', '$'), (8, 'a', '!'), (7, 'c', '@')]
>>> sorted(chars)
[(7, 'c', '@'), (8, 'a', '!'), (8, 'a', '$')]
```

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

Changing the Default Sorting Behavior

- To better understand the `sorted()` function, look at documentation

```
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.
```

- An ***iterable*** is any object over which we can iterate (list, string, tuple, range)
- The optional parameter **key** specifies a function that determines how each element should be compared to other elements
- The optional boolean parameter **reverse** (which by default is set to **False**) allows us to sort in reverse order
- Note: the `.sort()` list method also supports these options

Reverse Sorting

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

```
>>> fruits = [(12, 'apples'), (4, 'bananas'), (27, 'grapes')]
>>> sorted(fruits, reverse=True)
[(27, 'grapes'), (12, 'apples'), (4, 'bananas')]
>>> letters = ('a', 'c', 'z', 'b', 'Z', 'A')
>>> sorted(letters, reverse=True)
['z', 'c', 'b', 'a', 'Z', 'A']
>>> nums = (42, -20, 13, 10, 0, 11, 18)
>>> sorted(nums, reverse=True)
[42, 18, 13, 11, 10, 0, -20]
```

Sorting with a key Function

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: 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',   90, 'Spring'), ('CS136',   60, 'Spring'),
           ('AFR206',  30, 'Spring'), ('ECON233', 30, 'Fall'),
           ('MUS112',  10, 'Fall'),   ('STAT200', 50, 'Spring'),
           ('PSYC201', 50, 'Fall'),   ('MATH110', 90, '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()`**

```
# we 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

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

```python
courses = [('CS134',   90, 'Spring'), ('CS136',   60, 'Spring'),
           ('AFR206',  30, 'Spring'), ('ECON233', 30, 'Fall'),
           ('MUS112',  10, 'Fall'),   ('STAT200', 50, 'Spring'),
           ('PSYC201', 50, 'Fall'),   ('MATH110', 90, 'Spring')]
```

```python
# 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',   90, 'Spring'),
 ('MATH110', 90, 'Spring')]
```

Python Sorting is Stable

- Python's sorting functions are **`stable`**, which means that items that are equal according to the sorting **`key`** have the same relative order as in the original sequence

```
courses = [('CS134',   90, 'Spring'), ('CS136',   60, 'Spring'),
           ('AFR206',  30, 'Spring'), ('ECON233', 30, 'Fall'),
           ('MUS112',  10, 'Fall'),   ('STAT200', 50, 'Spring'),
           ('PSYC201', 50, 'Fall'),   ('MATH110', 90, 'Spring')]
```

```
def term(courseTuple):
    '''Takes a sequence and returns item at index 2'''
    return courseTuple[2]
```

```
sorted(courses, key=term)
```

```
[('ECON233', 30, 'Fall'),
 ('MUS112',  10, 'Fall'),
 ('PSYC201', 50, 'Fall'),
 ('CS134',   90, 'Spring'),
 ('CS136',   60, 'Spring'),
 ('AFR206',  30, 'Spring'),
 ('STAT200', 50, 'Spring'),
 ('MATH110', 90, 'Spring')]
```

Notice the ordering of courses with Fall term and those with Spring term

Takeaways

- Tuples are a new immutable sequence that
  - supports all sequence operations such as indexing and slicing
  - are useful for argument unpacking, multiple assignments
  - are useful for handling list-like data without aliasing issues
- `sorted()` function and `.sort()` list method sorts sequences in ascending and lexicographic order by default
- We can override the default sorting behavior by supplying optional parameters `key` (function), and `reverse` (Boolean)

The end!