LING 388: Computers and Language

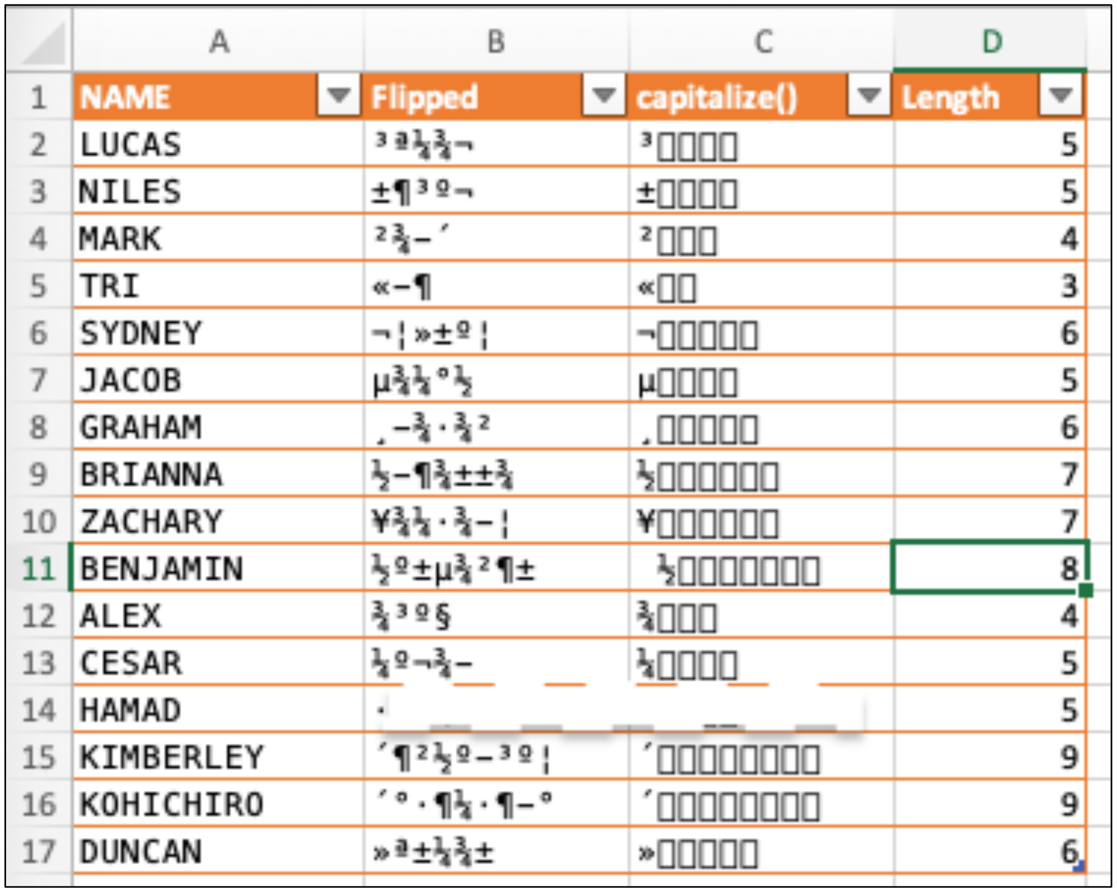
Lecture 6

Today's Topics

- Homework 4 review: *plus a bit extra*
- Python
  - strings vs. lists
  - `max()`
  - `sorted()`
  - `range()`
    - example of use in calculating compound interest
  - `sys.argv`
    - a special list: command line arguments
  - modifying lists
    - append, extend, insert, remove

Homework 4 Review

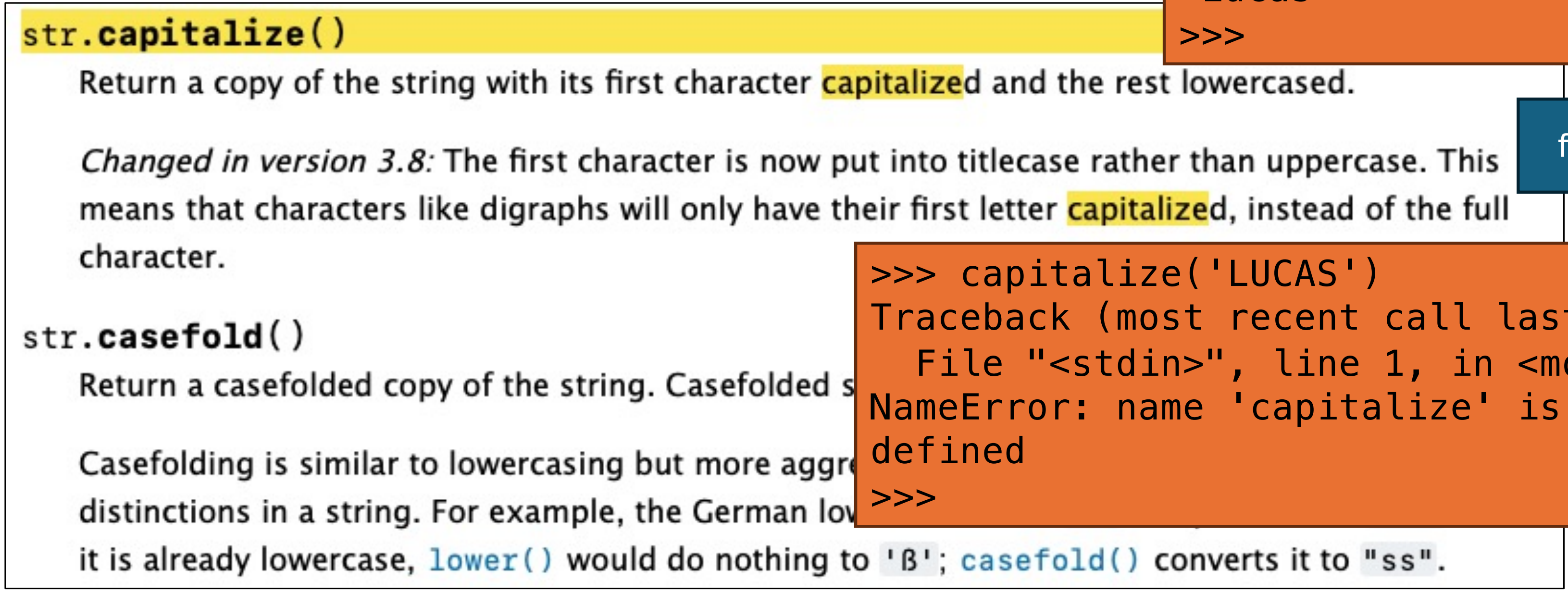
```
>>> flipname('JOHN')
JOHN (4) =>  Q1:
4a4f484e => Q2:
1246709838
0b1001010010011110100
100001001110 => Q3:
b5b0b7b1 => Q4: µ°·±
>>> flipname('John')
John (4) =>  Q1:
4a6f686e => Q2:
1248815214
0b1001010011011110110
100001101110 => Q3:
b5909791 => Q4: µ□□□
```



	A	B	C	D
1	NAME	Flipped	capitalize()	Length
2	LUCAS	³ª¼¾¬	³□□□□	5
3	NILES	±¶³º¬	±□□□□	5
4	MARK	²¾-´	²□□□	4
5	TRI	«-¶	«□□	3
6	SYDNEY	¬¦»±º¦	¬□□□□□	6
7	JACOB	µ¾¼°½	µ□□□□	5
8	GRAHAM	¸-¾·¾²	¸□□□□□	6
9	BRIANNA	½-¶¾±±¾	½□□□□□□	7
10	ZACHARY	¥¾¼·¾-¦	¥□□□□□□	7
11	BENJAMIN	½º±µ¾²¶±	½□□□□□□□	8
12	ALEX	¾³º§	¾□□□	4
13	CESAR	¼º¬¾-	¼□□□□	5
14	HAMAD	· [illegible]	[illegible]	5
15	KIMBERLEY	´¶²½º-³º¦	´□□□□□□□□	9
16	KOHICHIRO	´°·¶¼·¶-°	´□□□□□□□□	9
17	DUNCAN	»ª±¼¾±	»□□□□□	6

str.capitalize()

https://docs.python.org/3/library/stdtypes.html#str.capitalize



`str.capitalize()`

Return a copy of the string with its first character capitalized and the rest lowercased.

Changed in version 3.8: The first character is now put into titlecase rather than uppercase. This means that characters like digraphs will only have their first letter capitalized, instead of the full character.

`str.casefold()`

Return a casefolded copy of the string. Casefolded s

Casefolding is similar to lowercasing but more aggre
distinctions in a string. For example, the German lov
it is already lowercase, `lower()` would do nothing to `'ß'`; `casefold()` converts it to `"ss"`.

method

```
>>> 'LUCAS'.capitalize()
'Lucas'
>>>
```

function

```
>>> capitalize('LUCAS')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'capitalize' is not
defined
>>>
```

Homework 4 Review

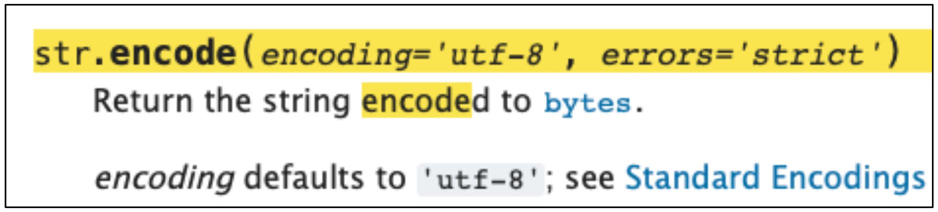
- Character by character:

```
>>> hex(ord('J'))
'0x4a'
>>> hex(ord('o'))
'0x6f'
>>> hex(ord('h'))
'0x68'
>>> hex(ord('n'))
'0x6e'
So 'John' is '4a6f686e'
```

Homework 4 Review

- Example:

```
>>> 'JOHN'.encode('ascii').hex()
'4a4f484e'
>>> 'JOHN'.encode('UTF-8').hex()
'4a4f484e'
>>> 'César'.encode('UTF-8').hex()
'43c3a9736172'
>>> 'César'.encode('ascii').hex()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeEncodeError: 'ascii' codec can't encode character '\xe9' in position 1:
ordinal not in range(128)
```



`str.encode(encoding='utf-8', errors='strict')`

Return the string encoded to `bytes`.

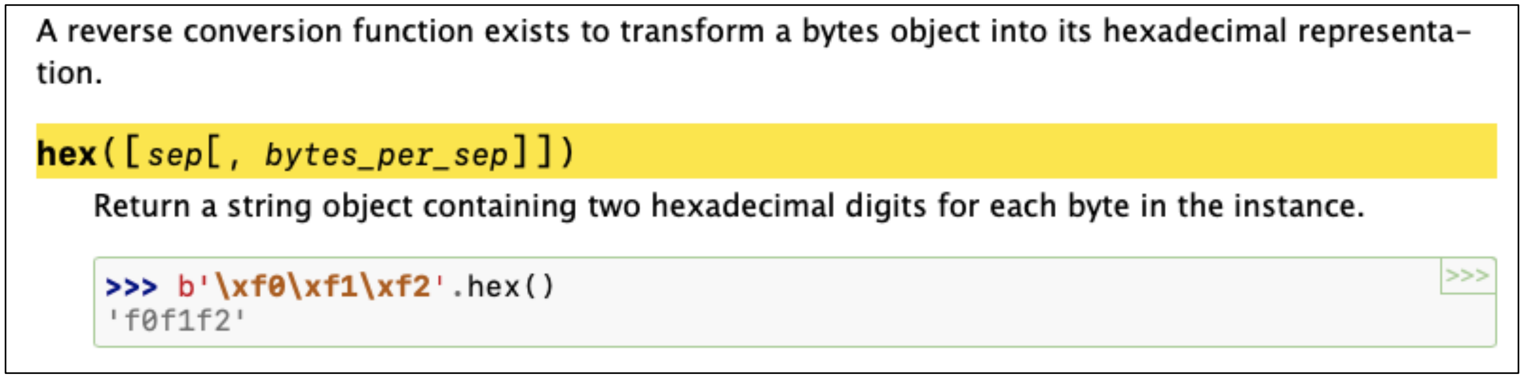
encoding defaults to `'utf-8'`; see Standard Encodings

Bytes literals are always prefixed with 'b' or 'B'; they produce an instance of the bytes type instead of the str type. They may only contain ASCII characters; bytes with a numeric value of 128 or greater must be expressed with escapes.

Homework 4 Review

- Example:

```
>>> 'JOHN'.encode('ascii').hex()
'4a4f484e'
>>> 'JOHN'.encode('UTF-8').hex()
'4a4f484e'
```



A reverse conversion function exists to transform a bytes object into its hexadecimal representation.

hex([*sep*[, *bytes_per_sep*]])

Return a string object containing two hexadecimal digits for each byte in the instance.

```
>>> b'\xf0\xf1\xf2'.hex()
'f0f1f2'
```

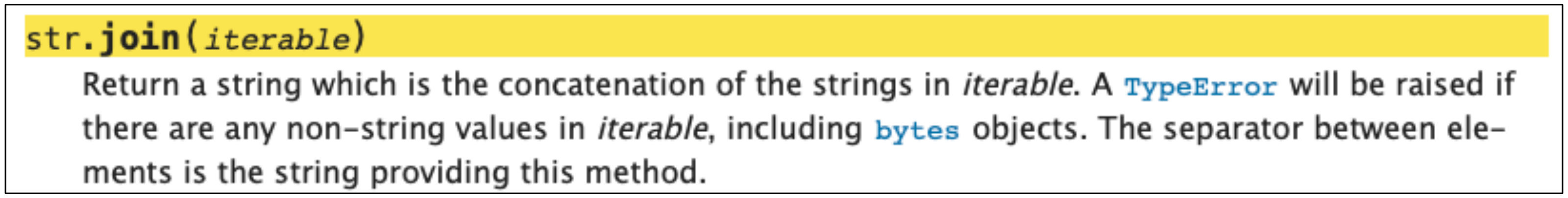
Homework 4 Review

```
>>> [hex(ord(x)) for x in 'John']
['0x4a', '0x6f', '0x68', '0x6e']
>>> ''.join(['{0:x}'.format(ord(x)) for x in 'John'])
'4a6f686e'
```

'J' 'o' 'h' 'n' *from* `for x in 'John'`

4a *from* `'{0:x}'.format(74)`

concatenate with no space between `''.join(list)`



`str.join(iterable)`

Return a string which is the concatenation of the strings in *iterable*. A `TypeError` will be raised if there are any non-string values in *iterable*, including `bytes` objects. The separator between elements is the string providing this method.

Homework 4 Review

- Example:

```
>>> x = 0x4a6f686e
>>> x
1248815214
>>> bin(0x4a6f686e)
'0b1001010011011110110100001101110'
>>> hex(0x4a6f686e ^ 0xffffffff)
'0xb5909791'
```

Question 2: decimal and binary

Question 3: flip!

- character by character:

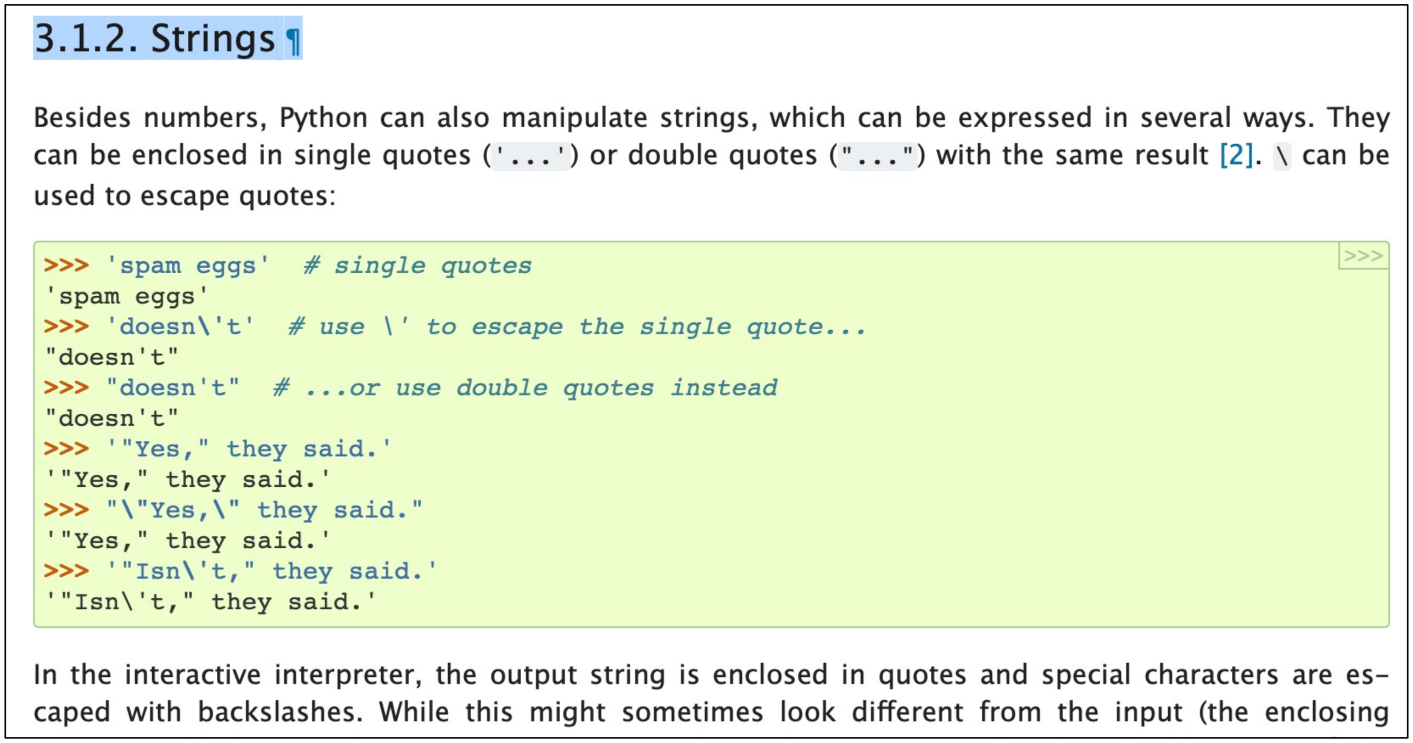
```
>>> chr(0xb5)
'µ'
…
```

Question 4: flipped name

- or:

```
>>> [chr(int(x,16)) for x in re.findall('..', 'b5909791')]
['µ', '\x90', '\x97', '\x91']
```

Python: Strings



3.1.2. Strings ¶

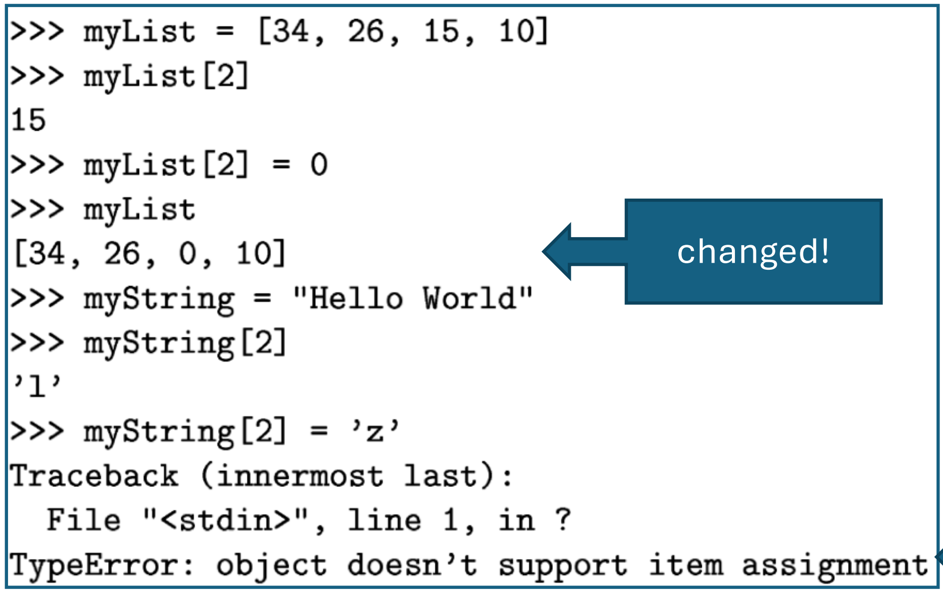
Besides numbers, Python can also manipulate strings, which can be expressed in several ways. They can be enclosed in single quotes (`'...'`) or double quotes (`"..."`) with the same result [2]. `\` can be used to escape quotes:

```
>>> 'spam eggs'  # single quotes
'spam eggs'
>>> 'doesn\'t'  # use \' to escape the single quote...
"doesn't"
>>> "doesn't"  # ...or use double quotes instead
"doesn't"
>>> '"Yes," they said.'
'"Yes," they said.'
>>> "\"Yes,\" they said."
'"Yes," they said.'
>>> '"Isn\'t," they said.'
'"Isn\'t," they said.'
```

In the interactive interpreter, the output string is enclosed in quotes and special characters are escaped with backslashes. While this might sometimes look different from the input (the enclosing

List vs. Strings

- Although Strings are like Lists, Lists are **mutable**, Strings are not.

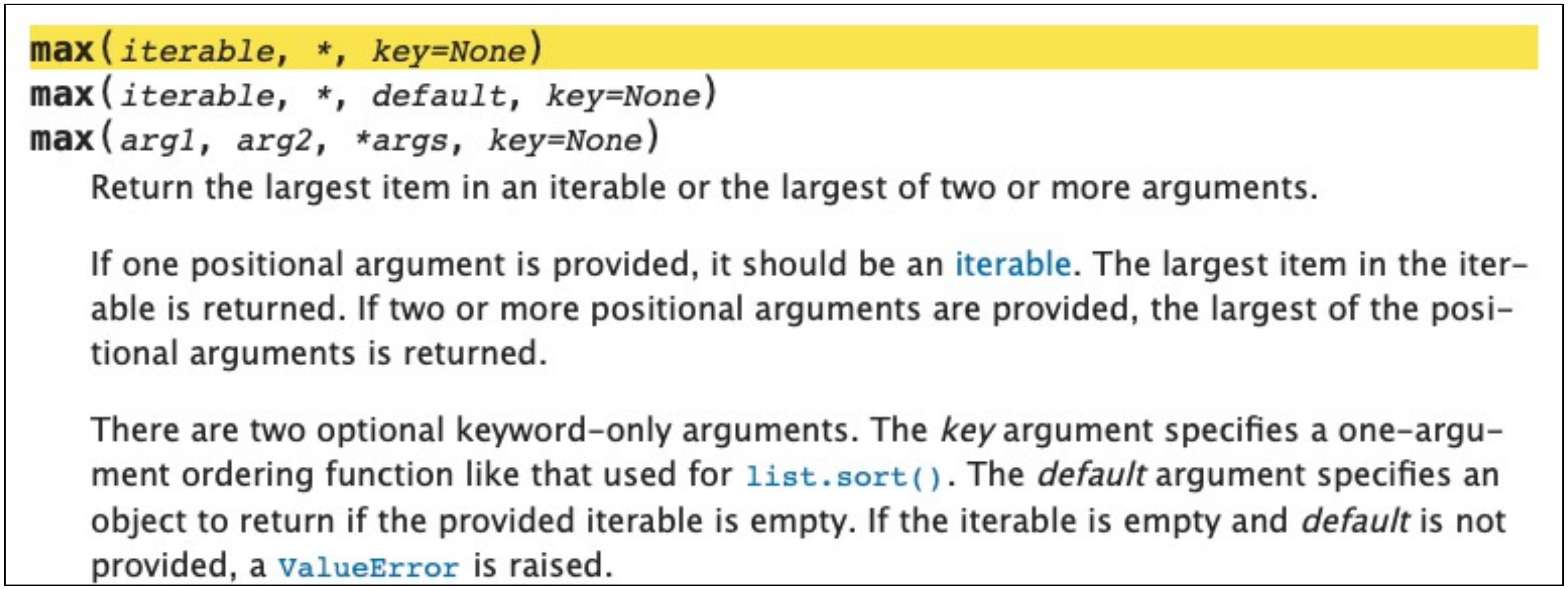


```
>>> myList = [34, 26, 15, 10]
>>> myList[2]
15
>>> myList[2] = 0
>>> myList
[34, 26, 0, 10]
>>> myString = "Hello World"
>>> myString[2]
'l'
>>> myString[2] = 'z'
Traceback (innermost last):
  File "<stdin>", line 1, in ?
TypeError: object doesn't support item assignment
```

changed!

not mutable!

max()



```
max(iterable, *, key=None)
max(iterable, *, default, key=None)
max(arg1, arg2, *args, key=None)
```

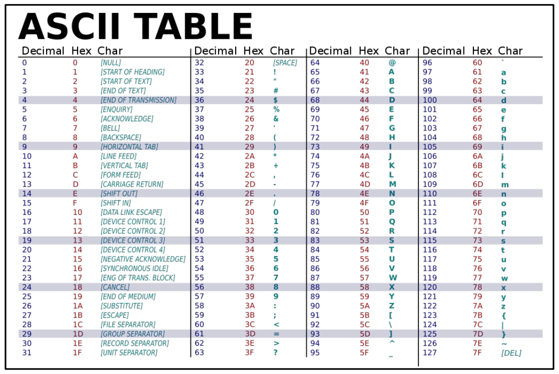
Return the largest item in an iterable or the largest of two or more arguments.

If one positional argument is provided, it should be an iterable. The largest item in the iterable is returned. If two or more positional arguments are provided, the largest of the positional arguments is returned.

There are two optional keyword-only arguments. The *key* argument specifies a one-argument ordering function like that used for `list.sort()`. The *default* argument specifies an object to return if the provided iterable is empty. If the iterable is empty and *default* is not provided, a `ValueError` is raised.

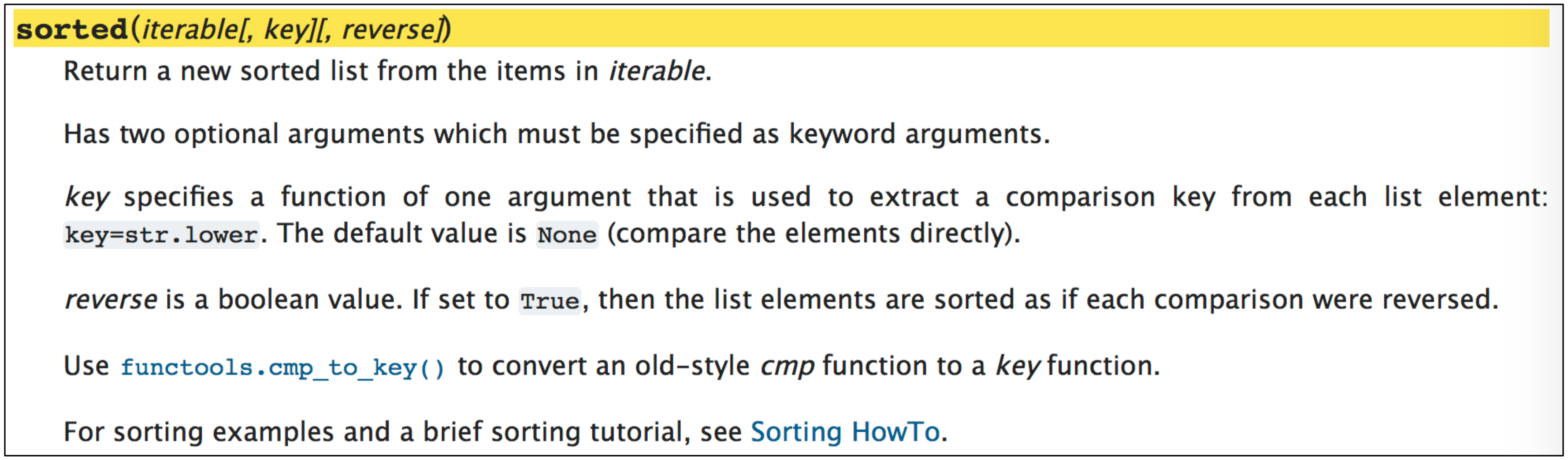
Python: sorting

sort order imposed historically by this table



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]



`sorted`(*iterable[, key][, reverse]*)

Return a new sorted list from the items in *iterable*.

Has two optional arguments which must be specified as keyword arguments.

key specifies a function of one argument that is used to extract a comparison key from each list element: `key=str.lower`. The default value is `None` (compare the elements directly).

reverse is a boolean value. If set to `True`, then the list elements are sorted as if each comparison were reversed.

Use `functools.cmp_to_key()` to convert an old-style *cmp* function to a *key* function.

For sorting examples and a brief sorting tutorial, see Sorting HowTo.

Python: `range()`

- `range()`: equivalent to producing a list of numbers (*sequence*)
  - `range(n)` — `[0, 1, .., n−1]` — **note**: excludes n!
  - `range(start,n)` — `[start, start+1, .., n−1]`
  - `range(start,n,step)` — `[start, start+step,…, last]`
    - `last: start + k ∗ step < n`
  - `range(n,stop,−step)` — *counts down to stop*

range(Start, End, Step) (integers only!)

```
>>> range(1,10,2)
range(1, 10, 2)
>>> list(range(1,10,2))
[1, 3, 5, 7, 9]
```

note: list() converts sequence into a list

Python: `range()`

```
python
>>> for odd in range(1,9,2):
...     print(odd*odd)
...
```

note: hit ENTER

```
1
9
25
49
```

note: 81 not printed!

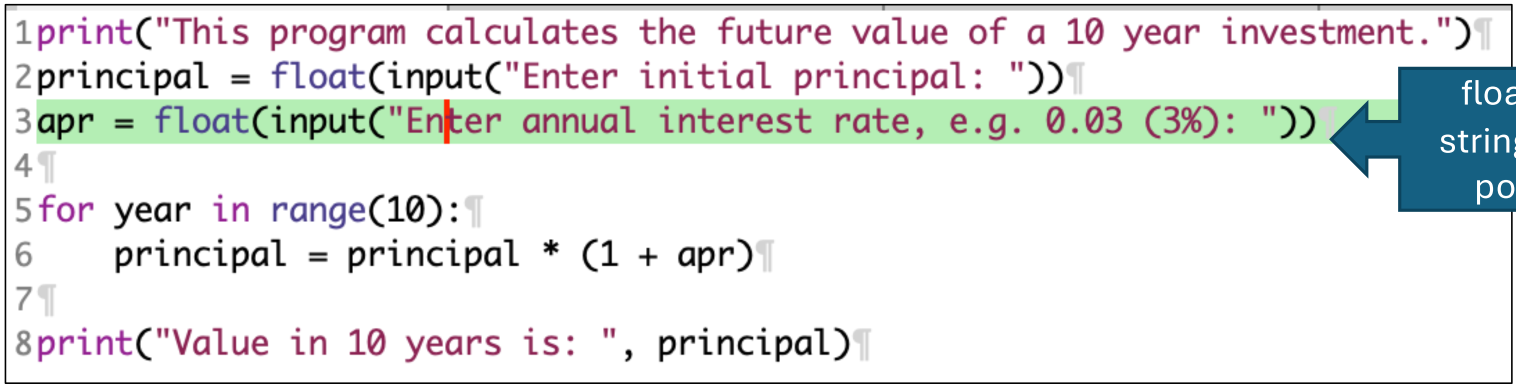
```
>>> for odd in range(9,1,2):
...     print(odd*odd)
...
>>> for odd in range(9,1,-2):
...     print(odd*odd)
...
81
49
25
9
```

note: 1 not printed!

```
>>>
```

Python Program: using range()

- File: `futval.py`

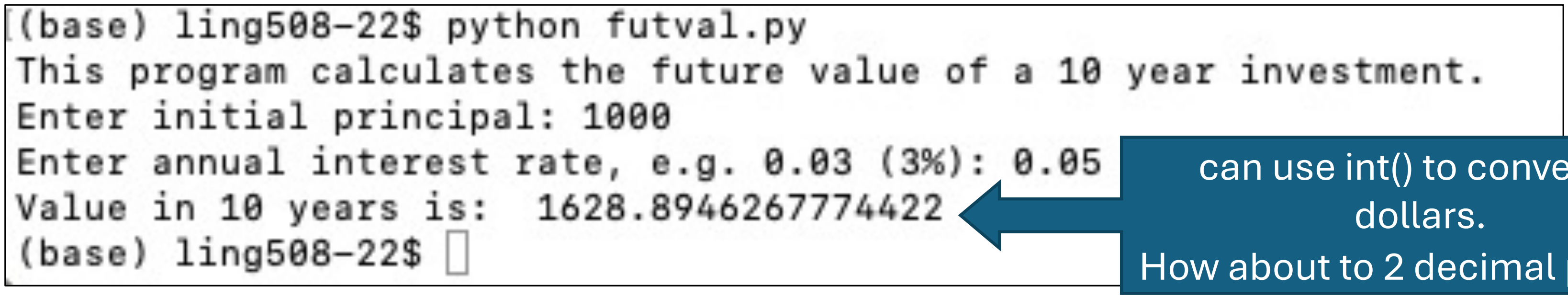


```
print("This program calculates the future value of a 10 year investment.")
principal = float(input("Enter initial principal: "))
apr = float(input("Enter annual interest rate, e.g. 0.03 (3%): "))

for year in range(10):
    principal = principal * (1 + apr)

print("Value in 10 years is: ", principal)
```

float() converts string to a floating point number

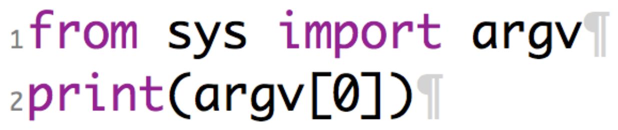


```
(base) ling508-22$ python futval.py
This program calculates the future value of a 10 year investment.
Enter initial principal: 1000
Enter annual interest rate, e.g. 0.03 (3%): 0.05
Value in 10 years is:  1628.894626777442
(base) ling508-22$
```

can use int() to convert to dollars.
How about to 2 decimal places?

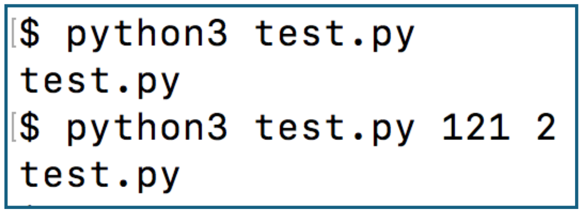
Python `sys.argv`

- List of arguments from the command line: what's `argv[0]` then?
- can use `len()` to calculate number of arguments



```
from sys import argv
print(argv[0])
```

test.py

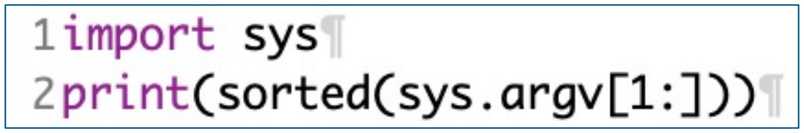


```
$ python3 test.py
test.py
$ python3 test.py 121 2
test.py
```

argv[1] and argv[2] ignored

Sorting on the command line (argv)

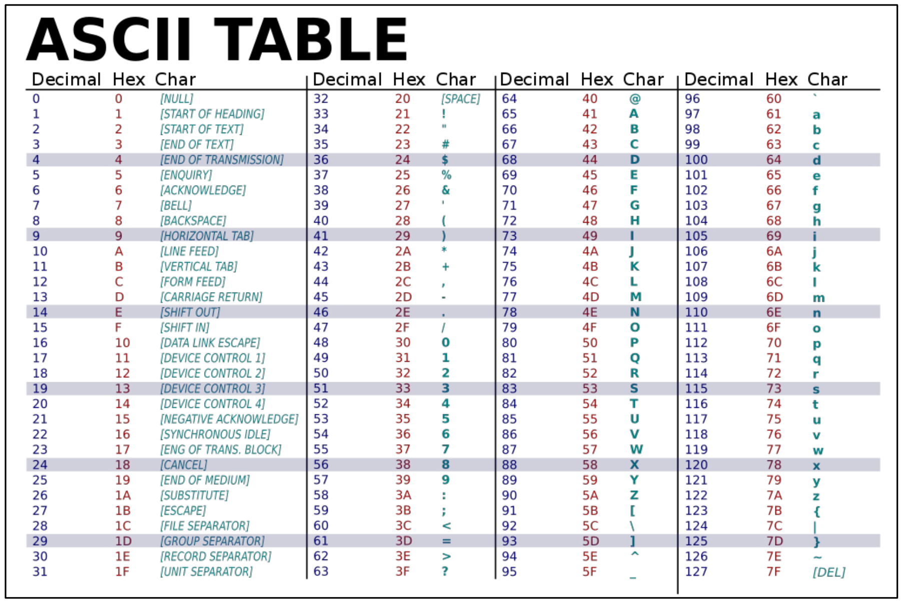
sort.py



```python
import sys
print(sorted(sys.argv[1:]))
```

```
$ python sort.py 20 50 30 9 1
['1', '20', '30', '50', '9']
```

notice these are strings!
not numbers



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]

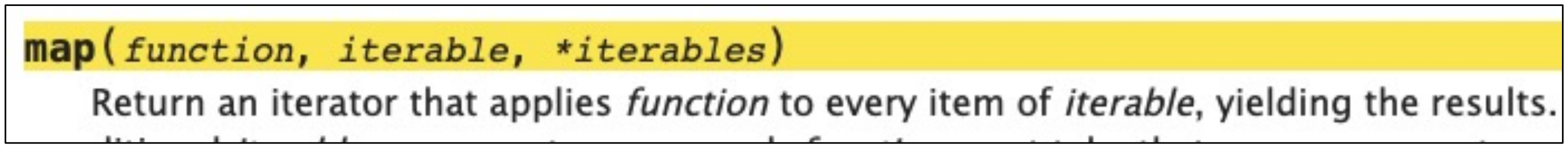
Sorting on the command line (argv)

- Numeric (sort2.py):

```
import sys
print(sorted(map(int, sys.argv[1:])))
```

- Run:

```
$ python sort2.py 20 50 30 9 1
[1, 9, 20, 30, 50]
```



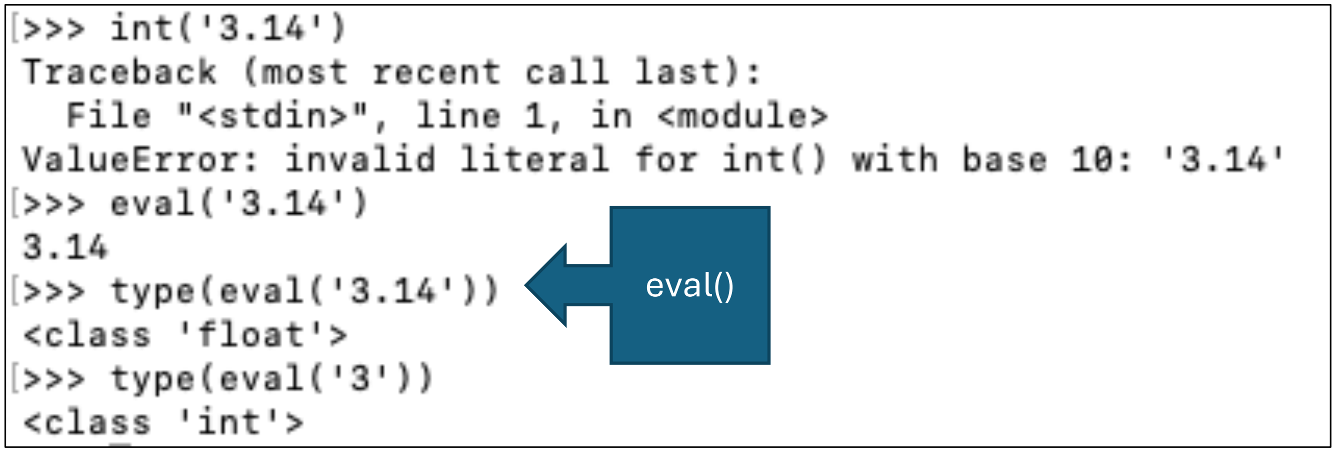
map(*function, iterable, *iterables*)

Return an iterator that applies *function* to every item of *iterable*, yielding the results.

Sorting on the command line (`argv`)

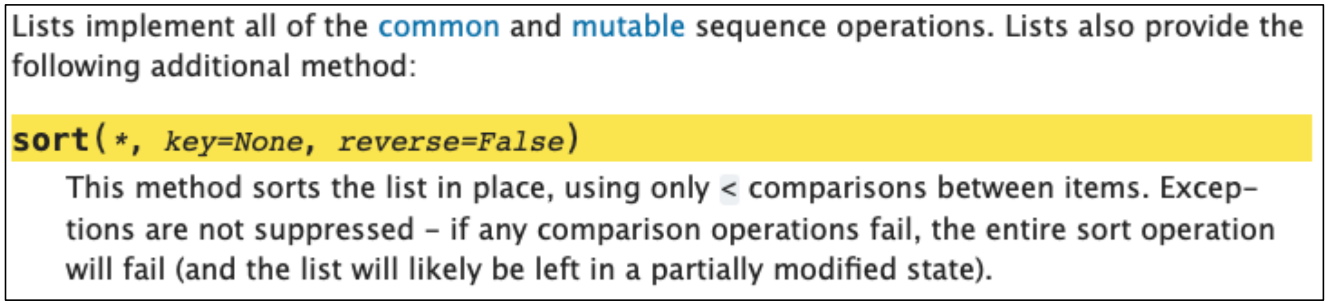
key = *function*:

- int: use function int()
- float: use function float() to convert into a floating point number.



```
>>> int('3.14')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '3.14'
>>> eval('3.14')
3.14
>>> type(eval('3.14'))
<class 'float'>
>>> type(eval('3'))
<class 'int'>
```

sorted() vs. *list*.sort()



Lists implement all of the common and mutable sequence operations. Lists also provide the following additional method:

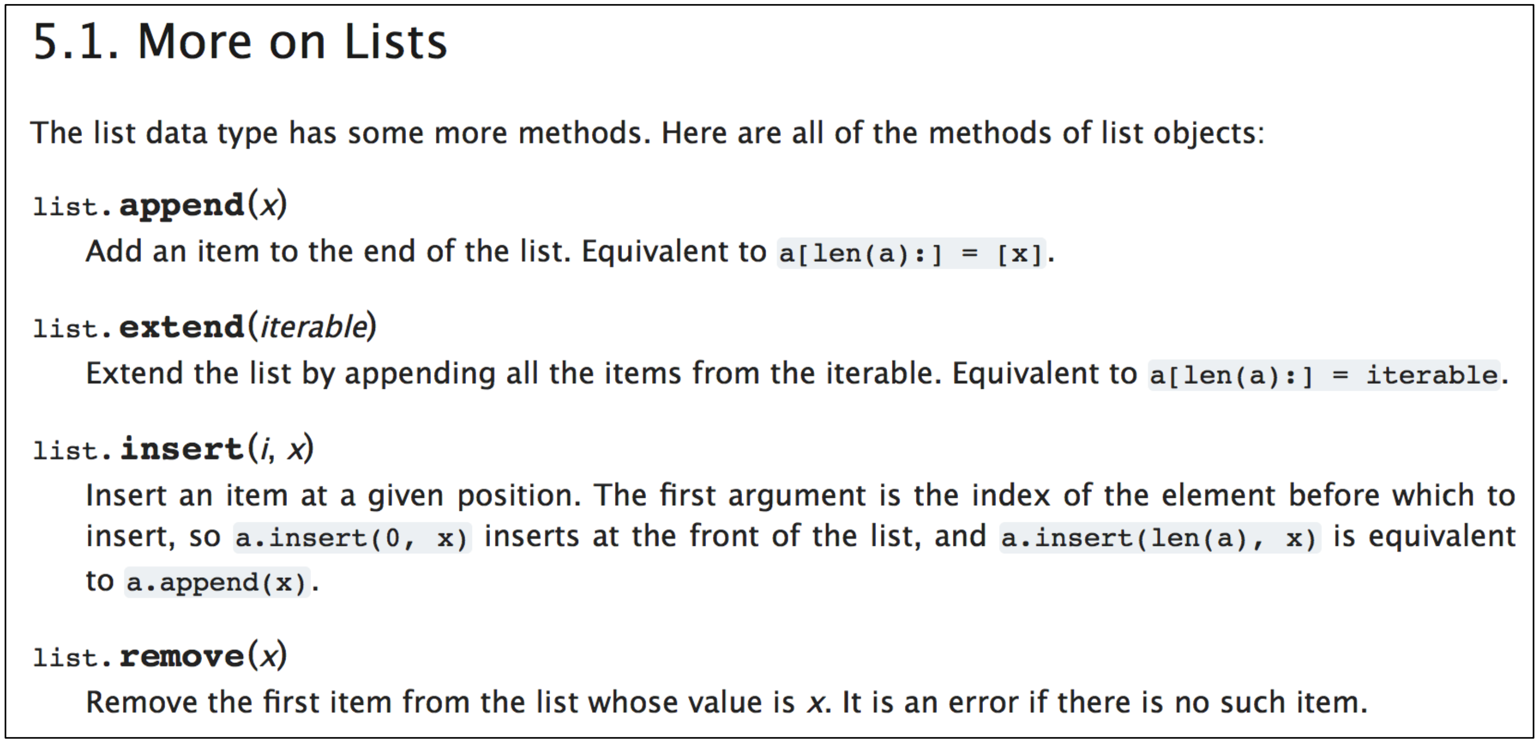
sort(**, key=None, reverse=False*)

This method sorts the list in place, using only `<` comparisons between items. Exceptions are not suppressed – if any comparison operations fail, the entire sort operation will fail (and the list will likely be left in a partially modified state).

- Example:

```
>>> list = ['jam', 'greedily', 'brook', 'sheep_', 'large', 'nap', 'twos', 'handkerchief', 'sulky', 'sawdust', 'count']
>>> sorted(list)
['brook', 'count', 'greedily', 'handkerchief', 'jam', 'large', 'nap', 'sawdust', 'sheep_', 'sulky', 'twos']
>>> list
['jam', 'greedily', 'brook', 'sheep_', 'large', 'nap', 'twos', 'handkerchief', 'sulky', 'sawdust', 'count']
>>> list.sort()
>>> list
['brook', 'count', 'greedily', 'handkerchief', 'jam', 'large', 'nap', 'sawdust', 'sheep_', 'sulky', 'twos']
```

Python Lists



5.1. More on Lists

The list data type has some more methods. Here are all of the methods of list objects:

`list.`**`append`**(*x*)
Add an item to the end of the list. Equivalent to `a[len(a):] = [x]`.

`list.`**`extend`**(*iterable*)
Extend the list by appending all the items from the iterable. Equivalent to `a[len(a):] = iterable`.

`list.`**`insert`**(*i*, *x*)
Insert an item at a given position. The first argument is the index of the element before which to insert, so `a.insert(0, x)` inserts at the front of the list, and `a.insert(len(a), x)` is equivalent to `a.append(x)`.

`list.`**`remove`**(*x*)
Remove the first item from the list whose value is *x*. It is an error if there is no such item.