Announcements

- Next Friday is a public holiday and ALL LABS ON FRIDAY 29th MARCH HAVE BEEN MOVED TO A MAKE-UP TIME
  - Please remember to check your MyTimetable schedule and attend your make-up lab.
  - If you have problems with your allocated time, please use MyTimetable to move to a different lab. Please don't email the course address - we will just ask you to use myTimetable
- Homework 3 is due at the end of this week
- Those students with labs in HN1.25 – please note that your labs have been moved to better rooms.
  - The details of the new rooms should be in your MyTimetable. Please make sure that you attend these labs and the correct location.
- Head-count at each lab is monitored – and attendance is very good – well done!
- Apologies about the difficulties with HW1 marks. This was a problem with our course systems and these have now been fixed.
- The Drop-In session this week will be held in N113 CSIT Building on **Thursday** 1-2pm

Lecture Roadmap

- Intro to Programming
- Variables
- Functions
  - The stack
  - Scope
- Flow control
  - `if`
  - `while`
  - `for`
- Strings
- Lists
- Tuples
- Dictionaries

Sequences have elements

- Strings and Lists are Sequences in Python
- hello_world = “Hello, world!”

	H	e	l	l	o	,		w	o	r	l	d	!
Element:	0	1	2	3	4	5	6	7	8	9	10	11	12
									….	-4	-3	-2	-1

- Negative indexes are completely legal syntax (and useful)

Strings (pt II)

COMP1730/COMP6730

Reading: Textbook chapter 8 : Alex Downey, *Think Python*, 2nd Edition (2016)

OR

Chapter 5 : Lubanovic, *Introducing Python*, 2nd Edition (2019)

But only up until section: *Search and Select*

Australian National University

Strings are **immutable**

- Once a string is assigned, it can only be changed by re-assigning the whole string.
- If we try to change an element, we get an error:

```
>>> greeting = 'Hello, world!'
>>> greeting[0] = 'J'
TypeError: 'str' object does not support item assignment
```

- If we want to change this character, we need to reassign the string:

```
>>> greeting = 'Hello, world!'
>>> new_greeting = 'J' + greeting[1:]
>>> new_greeting
'Jello, world!'
```

Downey (2015) *Think Python*, 2nd Ed. (Chapter 8)

Strings and the `in` operator

- The keyword `in` can be used as a Boolean operator to test if a substring appears in another word:

```
>>> 'a' in 'banana'
True
>>> 'seed' in 'banana'
False
```

Downey (2015) *Think Python*, 2nd Ed. (Chapter 8)

`in` with `for` - string traversal

- The `in` keyword can also be used with `for` to iterate through a string:

```
prefixes = 'JKLMNOPQ'
suffix = 'ack'

for letter in prefixes:
    print(letter + suffix)
```

Downey (2015) *Think Python*, 2nd Ed. (Chapter 8)

- Output:

```
Jack
Kack
Lack
Mack
Nack
Oack
Pack
Qack
```

Example: `in`, `for` and string traversal

- And this is useful, for example – define a function to find common letters in words:

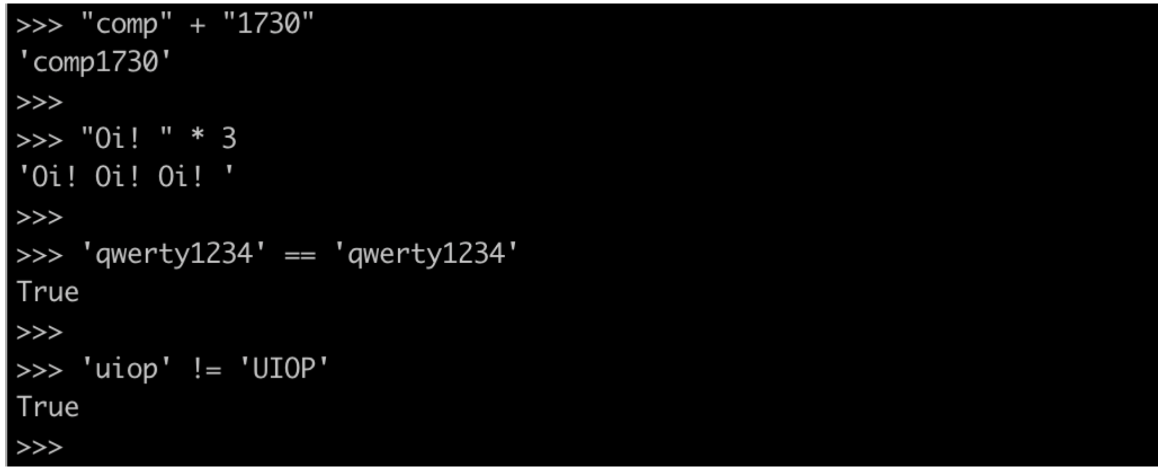
```
def in_both(word1, word2):
    for letter in word1:
        if letter in word2:
            print(letter)
```

```
>>> in_both('apples', 'oranges')
a
e
s
```

Downey (2015) *Think Python*, 2nd Ed. (Chapter 8)

Operations on sequences

- The type of a variable determines the meaning of operators applied to them:
  - On a `str`, the ‘+’ operator means concatenation
  - And the ‘*’ operator means repetition
  - == still tests for equality
  - != tests inequality



```
>>> "comp" + "1730"
'comp1730'
>>>
>>> "Oi! " * 3
'Oi! Oi! Oi! '
>>>
>>> 'qwerty1234' == 'qwerty1234'
True
>>>
>>> 'uiop' != 'UIOP'
True
>>>
```

Length of a string with `len()`

- Because a string is a sequence, we can use the sequence function len() to return the length of the sequence

```
>>> fruit = 'banana'
>>> len(fruit)
6
```

Downey (2015) *Think Python*, 2nd Ed. (Chapter 8)

- This function will return the length of any sequence – more later

Slicing to get sub-strings (Lubanovic Ch 5)

- Sometimes you will need to obtain a *substring* (part of a string)
- There is short-hand python syntax to make this easy - **slices**
- Because strings are **sequences**, you can get a substring by taking a **slice** of the sequence:

```
example_string[start:end]
```

-`start` is the index of the first element

-`end`

- Slicing works of all built-in sequence types (`str`, `list`, `tuple`) and returns the same type
- If `start` or `end` are left out, they default to the beginning and end (*ie.* after the last element)

Slices

- The slice range is ‘half-open’:
  - The element specified by the **start** index is *included*
  - But, the element specified by the **end** index is *left out*

```
>>> s = "Hello World"
>>> s[6:10]
'Worl'
```

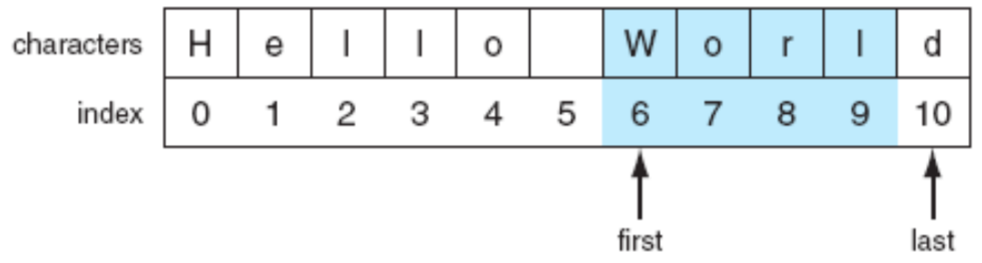


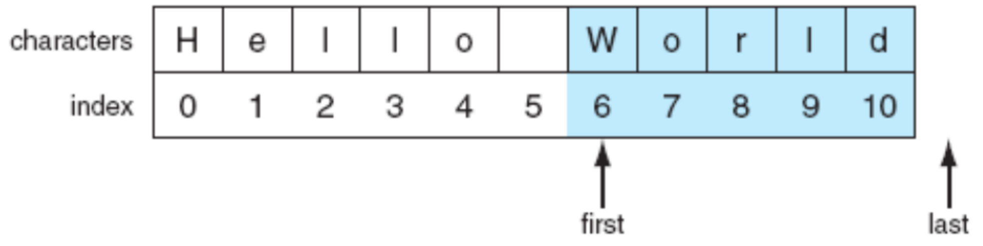
FIGURE 4.2 Indexing subsequences with slicing.

Punch & Enbody (2012) *The Practice of Computing using Python* (2nd ed.)

Slices

- If not specified explicitly, the `end` index defaults to the **last element** of the sequence

```
>>> s = "Hello World"
>>> s[6:]
'World'
```

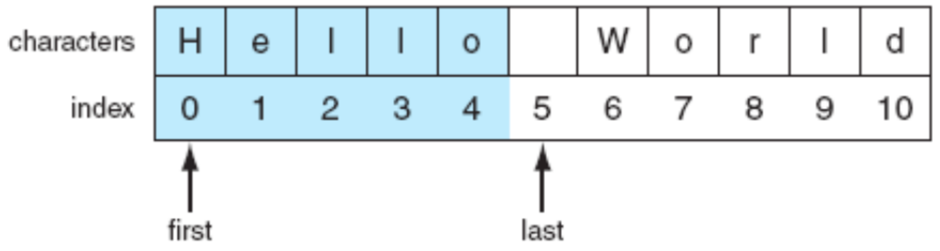


Punch & Enbody (2012) *The Practice of Computing using Python* (2nd ed.)

Slices

- The `start` index defaults to the beginning of the sequence:

```
>>> s = "Hello World"
>>> s[:5]
'Hello'
```



String methods: `upper()`

- Convert string to upper case letters:

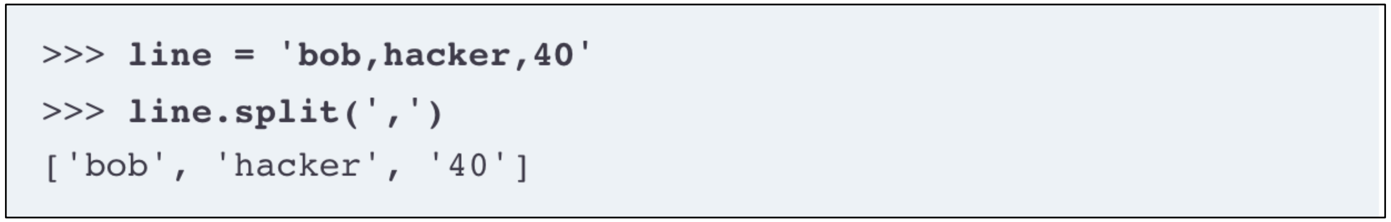
```
>>> word = 'banana'
>>> new_word = word.upper()
>>> new_word
'BANANA'
```

Downey (2015) Think Python, 2nd Ed.

- Notice again the use of ‘dot’ notation
- This is a string method – works only on `str` variables
- There is also the method `lower()`

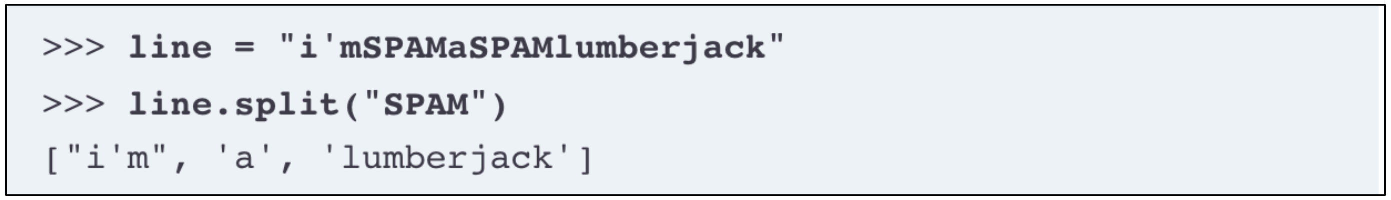
String methods: `split()`

- Splits a string at a delimiter, returns a List of resultant sub-strings.
- Comma-separated-values (CSV) is a common text data format. Split on commas:



```
>>> line = 'bob,hacker,40'
>>> line.split(',')
['bob', 'hacker', '40']
```

- But, you can use `split()` with any delimiter string:

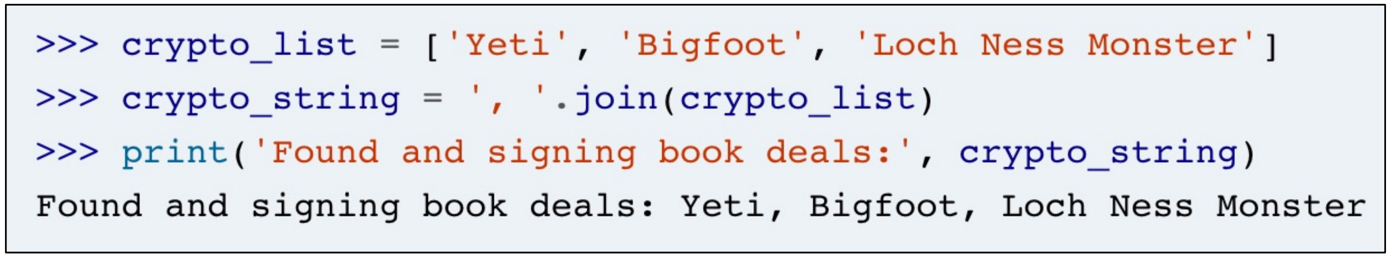


```
>>> line = "i'mSPAMaSPAMlumberjack"
>>> line.split("SPAM")
["i'm", 'a', 'lumberjack']
```

Lutz (2013) Learning Python, 5nd Ed.

String methods: `join()`

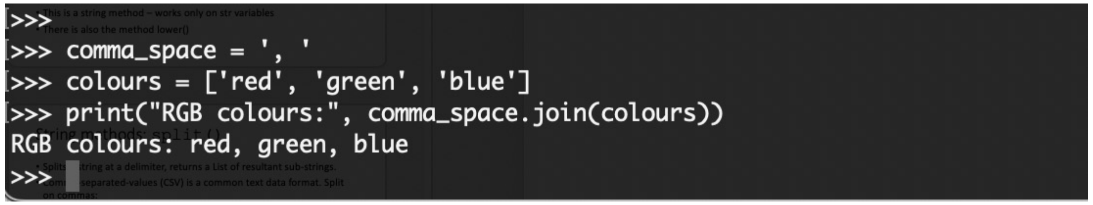
- The opposite of `split()`. Joins a List of strings, with a delimiter string:



```
>>> crypto_list = ['Yeti', 'Bigfoot', 'Loch Ness Monster']
>>> crypto_string = ', '.join(crypto_list)
>>> print('Found and signing book deals:', crypto_string)
Found and signing book deals: Yeti, Bigfoot, Loch Ness Monster
```

Lutz (2013) Learning Python, 5nd Ed.

- Note the use of the dot method on the string literal. This is a very python way of doing things.
- Alternatively, could also use a string delimiter variable with the dot notation:

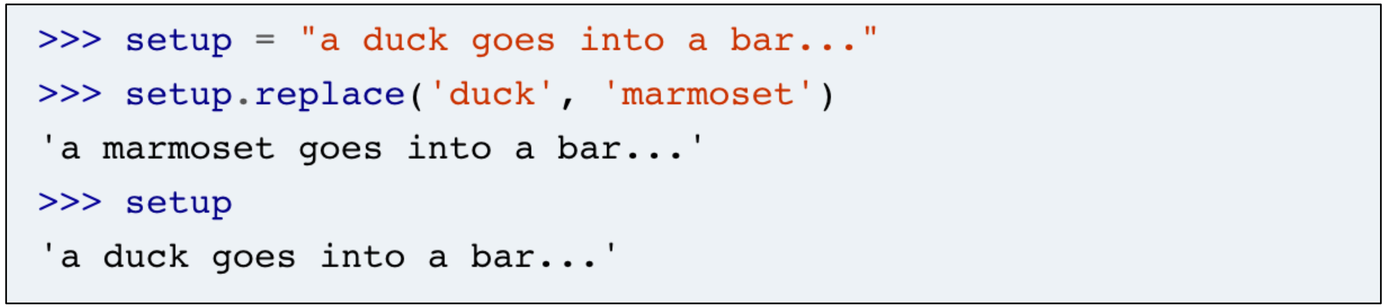


```
>>>
>>> comma_space = ', '
>>> colours = ['red', 'green', 'blue']
>>> print("RGB colours:", comma_space.join(colours))
RGB colours: red, green, blue
>>>
```

String methods: `replace()`

Australian National University

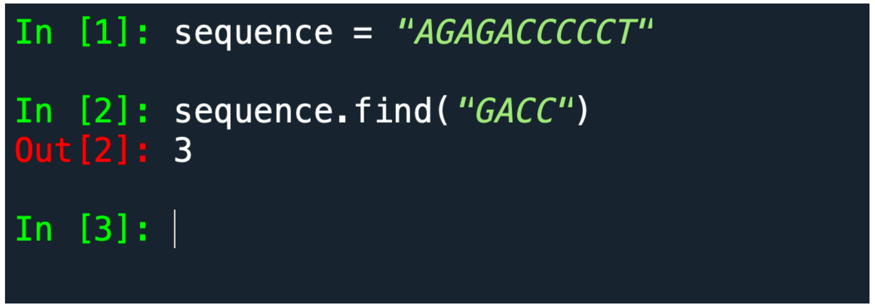
- Searches and replaces instances of a sub-string in a string variable:



```
>>> setup = "a duck goes into a bar..."
>>> setup.replace('duck', 'marmoset')
'a marmoset goes into a bar...'
>>> setup
'a duck goes into a bar...'
```

Lutz (2013) Learning Python, 5nd Ed.

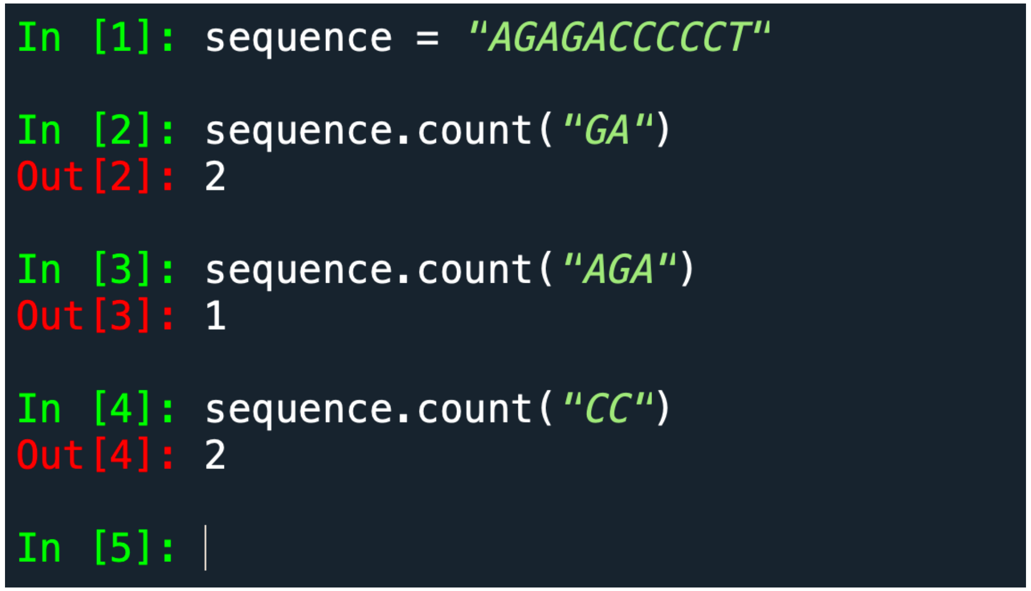
- There is also a similar `find()` method:
  - `find()` returns the lowest index position of a sub-string in a string variable.



```
In [1]: sequence = "AGAGACCCCCT"

In [2]: sequence.find("GACC")
Out[2]: 3

In [3]: 
```

String methods: `count()`

- The string method `count()` returns the count of the non-overlapping occurrences of another string:



```
In [1]: sequence = "AGAGACCCCCT"

In [2]: sequence.count("GA")
Out[2]: 2

In [3]: sequence.count("AGA")
Out[3]: 1

In [4]: sequence.count("CC")
Out[4]: 2

In [5]: 
```

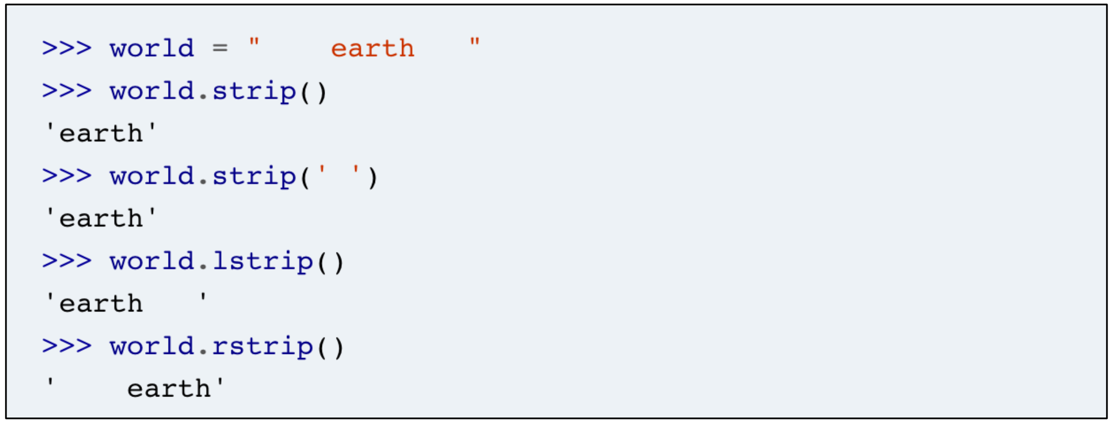
- For more information, try `help(str.count)`

Example: strings and string methods

- `str.count()` returns non-overlapping count
- Can we use `str.find()` in a function to return overlapping counts?

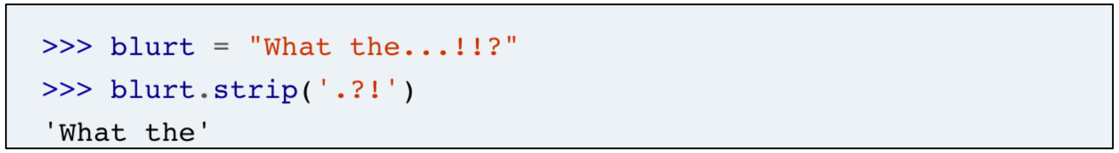
String methods: `strip(),lstrip(), rstrip()`

- In practice, parsed strings tend to have trailing spaces and newline characters. Use `strip(),lstrip()` and `rstrip()` to easily remove these:



```
>>> world = "    earth   "
>>> world.strip()
'earth'
>>> world.strip(' ')
'earth'
>>> world.lstrip()
'earth   '
>>> world.rstrip()
'    earth'
```

- And, it is possible to specify exactly what to trim:



```
>>> blurt = "What the...!!?"
>>> blurt.strip('.?!')
'What the'
```

Lutz (2013) Learning Python, 5nd Ed.

String methods: `format()`

- Inserting string variables into a pre-defined sentence is commonly useful. The string method `format()` makes this easy:

```
>>> thing = 'woodchuck'
>>> place = 'lake'
>>> 'The {} is in the {}.'.format(thing, place)
'The woodchuck is in the lake.'
```

Lubanovic (2019) Introducing Python, 2nd Ed.

- Note the used of the 'curly braces' {} to indicate where the the text the string variables should be inserted

Exercises

- Exercises 8-1, 8-2 and 8-4, *Think Python* Ch. 8
- Exercises in Lutz Ch 5 are a little different to what we've seen

Reading

- *Think Python* Ch 8

Lists (part I)

COMP1730/COMP6730

Reading: Textbook chapter 10 : Alex Downey, *Think Python*, 2nd Edition (2016)

Australian National University

Lists (finally) (*Think Python* Ch. 10)

Australian National University

- A list is a sequence. Very useful – essential! You will use these a lot.
- A sequence in python is a continuous series a values, called elements, that also have an index value (a number)
- Some lists:

```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
>>> numbers = [42, 123]
>>> empty = []
```

- In python, a list can contain a mixture of variable types – and may be nested:

```
['spam', 2.0, 5, [10, 20]]
```

Downey (2015) Think Python, 2nd Ed.

- Lists in python may contain other sequences. This is known as nesting.

Creating lists

- You can use different ways to create a list:

```
my_list = list()   # creates an empty list
my_list = list([1,2,3,4])  # creates a list with the list argument supplied
my_list = [1,2,3,4]  # the same thing
```

- Say, you want to perform an operation on the list at the same time:

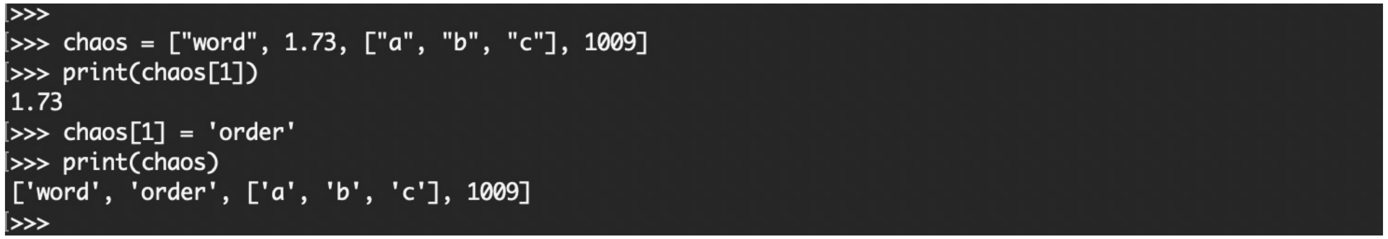
```
precise = [1.23, 1.99, 2.01, 2.51, 3.45]
rounded = []

for number in precise:
       rounded_number = round(number)
       rounded.append(rounded_number)
```

- `rounded` **becomes** `[1, 2, 2, 3, 3]`

Lists are mutable:

Australian National University

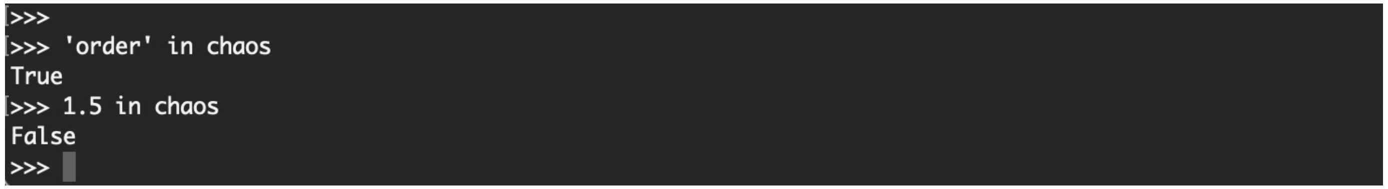
- The values of list elements can be changed:



```
>>>
>>> chaos = ["word", 1.73, ["a", "b", "c"], 1009]
>>> print(chaos[1])
1.73
>>> chaos[1] = 'order'
>>> print(chaos)
['word', 'order', ['a', 'b', 'c'], 1009]
>>>
```

Lists are sequences (and often work like strings):

- For instance, they have an `in` operator, like strings:



```
>>>
>>> 'order' in chaos
True
>>> 1.5 in chaos
False
>>>
```

Adding to a list with `append()`

- We can change the value held by existing elements. But we can't assign to an element that does not exist.
- To add elements onto the end of the list, we use the `append()` method:

```
>>> spam = ['cat', 'dog', 'bat']
>>> spam.append('moose')
>>> spam
['cat', 'dog', 'bat', 'moose']
```

- Or, we can insert into the middle of the list with `insert()`:

```
>>> spam = ['cat', 'dog', 'bat']
>>> spam.insert(1, 'chicken')
>>> spam
['cat', 'chicken', 'dog', 'bat']
```

Sweigart (2019) *Automate the boring stuff with python (Chapter 4)*

List operations

Australian National University

- Add lists together with '+' operator:

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> c
[1, 2, 3, 4, 5, 6]
```

- Multiply with '*' operator:

```
>>> [0] * 4
[0, 0, 0, 0]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Downey (2015) Think Python, 2nd Ed.

List traversal

Australian National University

- Like strings, lists can be traversed with a `for` loop:

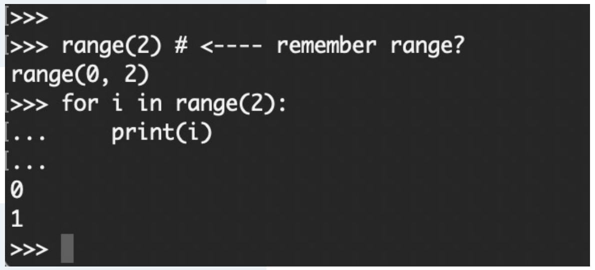
```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
```

```
for cheese in cheeses:
    print(cheese)
```

- And modified in the process, if desired:

```
>>> numbers = [42, 123]
```

```
for i in range(len(numbers)):
    numbers[i] = numbers[i] * 2
```



```
>>>
>>> range(2) # <---- remember range?
range(0, 2)
>>> for i in range(2):
...     print(i)
...
0
1
>>>
```

Downey (2015) Think Python, 2nd Ed.

List methods: `sort()`

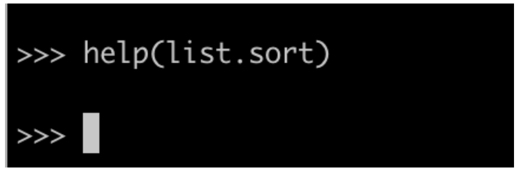
Australian National University

- Sort a list with `sort()`

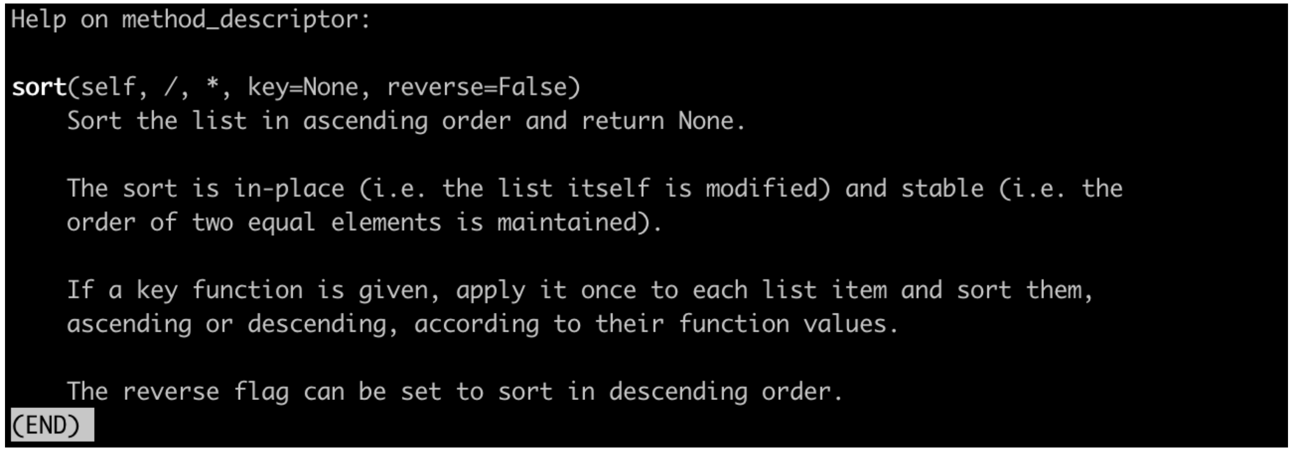
```
>>> t = ['d', 'c', 'e', 'b', 'a']
>>> t.sort()
>>> t
['a', 'b', 'c', 'd', 'e']
```

Downey (2015) Think Python, 2nd Ed. (chapter 10)

- Note how the sort is performed on the original list. The result is that the original list is sorted – and does not create a new list.



```
>>> help(list.sort)

>>> 
```



```
Help on method_descriptor:

sort(self, /, *, key=None, reverse=False)
    Sort the list in ascending order and return None.

    The sort is in-place (i.e. the list itself is modified) and stable (i.e. the
    order of two equal elements is maintained).

    If a key function is given, apply it once to each list item and sort them,
    ascending or descending, according to their function values.

    The reverse flag can be set to sort in descending order.
(END)
```

Deleting list elements: `pop()`

- Lists are mutable, but how to delete an element? With `pop()`.

```
>>> t = ['a', 'b', 'c']
>>> x = t.pop(1)
>>> t
['a', 'c']
>>> x
'b'
```

Downey (2015) Think Python, 2nd Ed. (chapter 10)

- The elements with higher indices all shuffle down one, to fill the gap left by the deleted element.
- There are other ways to delete elements, too: the `del` and `remove()` methods. Each with useful features.

Delete by value with `remove()`

- `pop()` deletes whatever value is present at the index specified.
- `remove()` deletes the first occurrence of a particular value:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam.remove('bat')
>>> spam
['cat', 'rat', 'elephant']
```

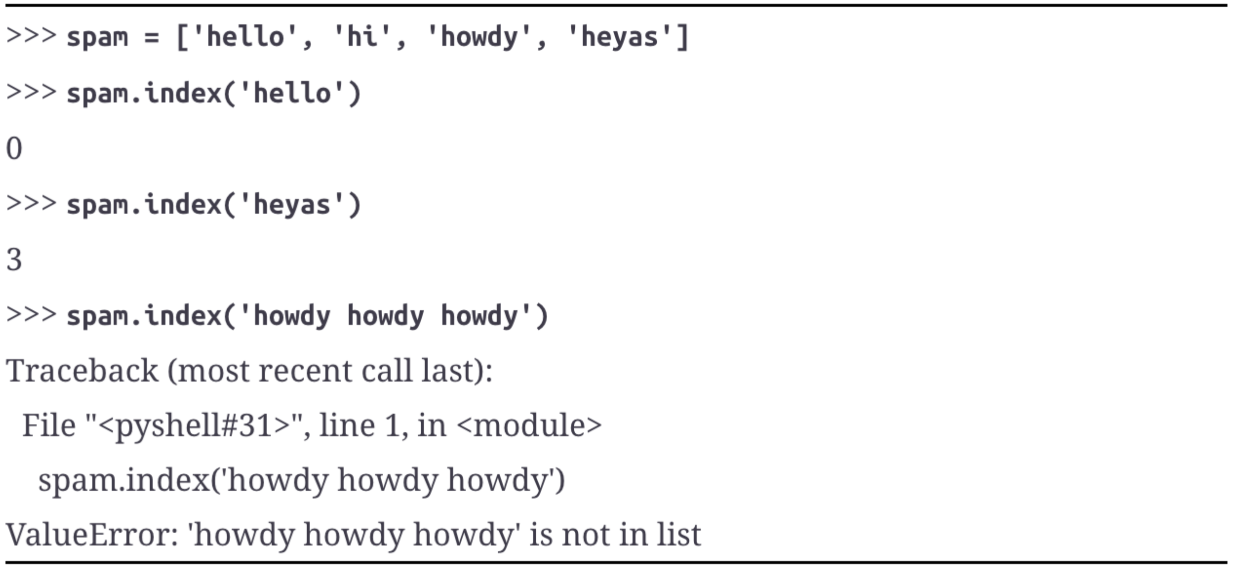
Sweigart (2019) *Automate the boring stuff with python (Chapter 4)*

- It won’t remove further occurrences of the value from the list
- You will also get a `ValueError` error if the list doesn’t contain the value specified

Searching a list with `index()`

Australian National University

- When you pass a value to the list method index(), it will return the index value of that value in the list:



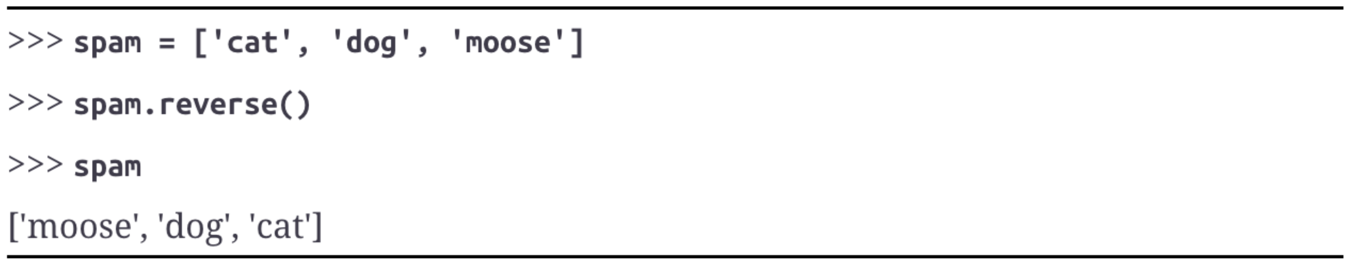
```
>>> spam = ['hello', 'hi', 'howdy', 'heyas']
>>> spam.index('hello')
0
>>> spam.index('heyas')
3
>>> spam.index('howdy howdy howdy')
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    spam.index('howdy howdy howdy')
ValueError: 'howdy howdy howdy' is not in list
```

Sweigart (2019) *Automate the boring stuff with python (Chapter 4)*

- Though, if the value isn't present you will get a `ValueError` error

`reverse()`

- Seemingly trivial, but `reverse()` is useful:



```
>>> spam = ['cat', 'dog', 'moose']
>>> spam.reverse()
>>> spam
['moose', 'dog', 'cat']
```

Sweigart (2019) *Automate the boring stuff with python (Chapter 4)*

More list methods

- Full list at https://docs.python.org/3/tutorial/datastructures.html

Method	Description
list.**append**(*x*)	Add an item to the end of the list.
list.**extend**(*iterable*)	Extend the list by appending all the items from the iterable.
list.**insert**(*i*, *x*)	Insert an item at a given position.
list.**remove**(*x*)	Remove the first item from the list whose value is equal to *x*.
list.**pop**([*i*])	Remove the item at the given position in the list,
list.**clear**()	Remove all items from the list.
list.**index**(*x*[, *start*[, *end*]])	Return zero-based index in the list of the first item whose value is equal to *x*.
list.**count**(*x*)	Return the number of times *x* appears in the list.
list.**sort**(**, key=None, reverse=False*)	Sort the items of the list in place
list.**copy**()	Return a shallow copy of the list.

Exercises

- Exercises 10-1, 10-3 and 10-4, *Think Python* Ch. 10

Reading

- *Think Python* Ch 10