COMP10001 Foundations of Computing
Iteration and Lists

Semester 1, 2019
Tim Baldwin, Nic Geard, Farah Khan, and Marion Zalk

— version: 1496, date: March 28, 2019 —

Announcements

- First **Revision Lecture** tomorrow (the first Advanced Lecture will be postponed to after the mid-semester test)
  - same time/location as usual, will cover material that has previously been covered in lectures, and will be recorded
  - come prepared with questions!
- Grok Worksheets 5–7 due end of Monday
- Mid-semester test at usual lecture time next Fri (5 April)
  - details of venues etc. to be posted on LMS
  - Thu lecture next week to be spent going over practice test
  - all lecture content to end of Week 4 (this week) is examinable
- Project 1 marking sheet on LMS

Lecture Agenda

- Last lecture – Grok Worksheet 8
  - Tuples and iteration
- This lecture – Grok Worksheets 8–9
  - Iteration (cont.)
  - Lists

Lecture Outline

Simple Examples

```
for i in range(0, 5):
    print(i, end='')
```


Simple Examples

```
mylen = 0
for i in "abc":
    mylen = mylen + 1
print(mylen)
```

is equivalent to:

```
mylen = 0
mylen = mylen + 1  # i = 'a'
mylen = mylen + 1  # i = 'b'
mylen = mylen + 1  # i = 'c'
print(mylen)
```

Simple Examples

- A more interesting example:

```
vowels = 0
word = "rhythm"
for char in  word:
    if char in "aeiou":
        vowels = vowels + 1
print(vowels)
```


Simple Examples

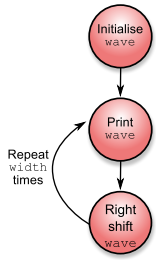
- Or alternatively with `range()`:

```
vowels = 0
word = "rhythm"
for i in range(len(word)):
    if word[i] in "aeiou":
        vowels = vowels + 1
print(vowels)
```

Particularly useful when you need to be able to recover *where* matches occurred

`for` Loop Practice: Mexican Wave

- Given the string `wave` made up of a `"Y"` and `width-1` repeats of `"x"`, how can we use a `for` loop to move the `"Y"` across one position to the right at a time?



Iteration: `while` Loops

- Another way to end `while` loops (and bypass the condition in the `while` statement) is via a `break` in the block of code

```
text = ''
while True:
    text = input('Enter 3-digit code: ')
    if len(text) == 3:
        break
    print('Sorry, invalid code.')
```

This prematurely and unconditionally exits from the loop

Iteration: `while` Loops

- A conditional loop:
  - The general idea is that we continue repeating a block of code as long as a given condition holds
  - Basic form:

```
while␣<condition>:
␣␣␣␣statement␣block
```

  - We use the notion of "block" as in `if` statements, but here, potentially the code block is repeated:

```
text = ''
while len(text) != 3:
    text = input('Enter 3-digit code: ')
```


Class Exercise

- If the sum of the digits of 4444^{4444} is A, and the sum of the digits of A is B, what is the sum of the digits of B?

Choosing between `for` and `while`

- If you need to iterate over all items of an iterable, use a `for` loop
- If there is a well defined end-point to the iteration which doesn't involve iterating over all items, use a `while` loop
- With a `for` loop, avoid modifying the object you are iterating over within the block of code
- Given a choice between the two, `for` loops are generally more elegant/safer/easier to understand

Lecture Outline

Lists: Mutable Data Type

- Lists are just like tuples but:
  - they are mutable
  - we use `[ , ]` rather than `( , )` to build them

```
>>> ["head", "tail", "tail"]
>>> [5, 5, 30, 10, 50]
>>> [1, 2, "buckle my shoe", 3.0, 4.0]
```

- As with all types, we can assign a list to a variable:

```
>>> fruit = ["orange", "apple", "apple"]
```


List Indexing and Splitting

- To access the items in a list we can use indexing (just like we do with strings and tuples):

```
>>> stuff_list = ["12", 23, 4, 'burp']
>>> stuff_list[-1]
'burp'
```

- We can similarly slice a list:

```
>>> stuff_list[:2]
['12', 23]
```

and calculate the length of a list with `len`

```
>>> len(stuff_list)
4
```


List Mutation

- Unlike tuples and strings, we can modify the internals of lists directly, either via assignment:

```
>>> stuff_list = ["12", 23, 4, 'burp']
>>> stuff_list[0] = '21'
>>> stuff_list
['21', 23, 4, 'burp']
>>> stuff_list[:2] = ['boing!']
>>> stuff_list
['boing!', 4, 'burp']
```

- ... or via methods that modify the internals of the list ...

List Mutation

- `append()` = add (single) item to end of list

```
>>> l = [1, 2, 3]
>>> l.append(4)
>>> l
[1, 2, 3, 4]
```

- `remove()` = remove the first instance of a given value

```
>>> l.remove(1)
>>> l
[2, 3, 4]
```

List Mutation

- `pop()` = remove the element of the given index given value

```
>>> l = [1, 2, 3]
>>> a = l.pop(2)
>>> b = l.pop(0)
>>> a, b, l
(3, 1, [2])
```

- `sort()` = sort the contents of the list (in-place)

```
>>> l = [4, 1, 3, 2]
>>> l.sort()
>>> l
[1, 2, 3, 4]
```


Mutability

- Types in Python can be either:
  - "immutable": the state of objects of that type cannot be changed after they are created
  - "mutable": the state of objects of that type **can** be changed after they are created
- Quiz:
  - Are strings mutable?
  - Are ints and floats mutable?

But What's the Difference?

- It seems that tuples and lists are the same, why have both?
- Important difference: mutability

```
>>> mylist = [1,2,3]
>>> mytuple = (1,2,3)
>>> mylist[1] = 6 ; print(mylist)
[1,6,3]
>>> mytuple[1] = 6 ; print(mytuple)
TypeError: 'tuple' object does not support ...
```


Lecture Summary

- What is a `for` loop?
- What is a `while` loop?
- What is a list?
- What are mutable types?