COMP10001 Foundations of Computing
Lists, Mutability and Function Debugging

Semester 2, 2018

Chris Leckie & Nic Geard

THE UNIVERSITY OF MELBOURNE

Announcements

- Workshops 5 & 6 due 23:59pm Monday 20/8
- Practice Project has been released on Grok. Note that this project is **not** assessed, so you can discuss code on the forums, etc.
- Project 1 will be released this Friday 17 August – Chris will be along to talk about it

Lecture Agenda

- Last lecture:
  - Functions
  - Iteration
- This lecture:
  - Lists
  - Mutability
  - How to "debug"?

Iteration recap: while and for loops

- `while` loops: repeat while a condition is true:

```
>>> a = 0
>>> while a < 5:
...     print(a, end=" ")
...     a = a + 1
0 1 2 3 4
```

- `for` loops: repeat for each element in a sequence:

```
>>> for i in (0, 1, 2, 3, 4):
...     print(i, end=" ")
0 1 2 3 4
```

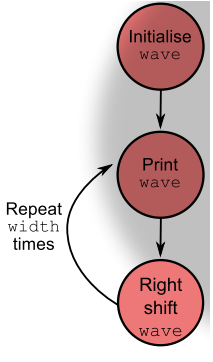
A Useful Function for creating a sequence

- `range(start=0,end,step=1)`: generate a sequence of `int` values from `start` (inclusive) to `end` (non-inclusive), counting `step` at a time

```
>>> for i in range(5):
...     print(i, end=" ")
0 1 2 3 4
>>> for i in range(0,10,2):
...     print(i, end=" ")
0 2 4 6 8
>>> for i in range(10,0,-1):
...     print(i, end=" ")
10 9 8 7 6 5 4 3 2 1
```

`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?



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

Class Exercise

- Assuming an unlimited number of coins of each of the following denominations:

  $$(1, 2, 5, 10, 20)$$

  calculate the number of distinct coin combinations which make up a given amount N (in cents).

Lists: An Introduction

- To date, we have discussed data types for storing single values (numbers or strings), and tuples for storing multiple things. There is another way to store multiple things: a "list".

```
["head","tail","tail"] # list of strings
[5,5,30,10,50] # list of ints
[1,2,"buckle my shoe",3.0,4.0] # allsorts
```

- 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):

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

- We can similarly slice a list:

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

and calculate the length of a list with `len()`

```
>>> len(listOfStuff)
4
```

Class Exercise

- Write code to extract the middle element from the list `l`:

```
>>> l = [1,2,3]
>>> middle(l)
[2]
>>> l = [1,2]
>>> middle(l)
[]
```

- What are the values of `l1` and `l2` after execution of the following code:

```
l1 = [1,2,3,4]
l2 = l1[::-1]
```

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 item assignment
```

- Tuples are immutable - they cannot be changed once created
- Lists are mutable - individual elements can be changed

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 lists mutable?
- Are tuples mutable?

Function Arguments I

A key place where mutability is important is when passing arguments to functions.

```
def f(l):
    l[1] = 6

mylist = [1,2,3,4,5]
f(mylist)
print(mylist)

mytuple = (1,2,3,4,5)
f(mytuple)
print(mytuple)
```

Function Arguments II

```
def f(l):
    if type(l) is list:
        l = l + [6]
    else:
        l = l + (6,)

mylist = [1,2,3,4,5]
f(mylist)
print(mylist)

mytuple = (1,2,3,4,5)
f(mytuple)
print(mytuple)
```

Function Arguments III

```
def f(l):
    if type(l) is list:
        l.append(6)
    else:
        l = l + (6,)
    return(l)

mylist = [1,2,3,4,5]
list2 = f(mylist)
print(mylist) ; print(list2)

mytuple = (1,2,3,4,5)
tuple2 = f(mytuple)
print(mytuple) ; print(tuple2)
```

Local Variables and Mutability I

- When you pass a mutable object to a function and locally mutate it in the function, the change is preserved in the global object:

```
def changeList(lst):
    lst = []
    return lst
def changeListItem(lst):
    lst[0] = "Changed, hah!"
```

```
>>> mylist = [1,2,3]
>>> changeList(mylist)
[]
>>> mylist
[1, 2, 3]
>>> changeListItem(mylist)
>>> mylist
['Changed, hah!', 2, 3]
```

Local Variables and Mutability II

- In fact, there is nothing specific to functions going on here; it is consistent with the behaviour of mutable objects user assignment/mutation:

```
>>> list1 = [1,2,3]
>>> list2 = list1
>>> list2[0] = "Changed, hah!"
>>> list2
['Changed, hah!', 2, 3]
>>> list1
['Changed, hah!', 2, 3]
```

Python Tips: Placement of Function Definitions

- Functions in Python must be defined before they are called (i.e. the definition must precede any code that calls them)
- This may seem curious, until you realise that function names are just variables, and the behaviour is identical to that of other variables

Function Practice

- A "pangram" is a string that contains all the letters of the English alphabet at least once, for example:

  `"The quick brown fox jumps over the lazy dog"`

  Write a function `pangram` to check whether a string `gram` is a pangram or not.

Function Debugging Practice

- What is wrong with the following, and how would we fix it?

```
def last_vowel(word):
    """Find the index of the last vowel
    in 'word'"""
    for i in range(len(word)):
        if word[i] in "aeiou":
            print(i)
        else:
            print(None)
```

- A great visualisation tool: `http://www.pythontutor.com` (not in Safari)