Mutable and Immutable

In python, everything is an object

Unlike C++ you have only objects and bindings (think vars and pointers)
Bindings act differently depending on mutability

The following are **immutable** objects:

- Numeric types: int, float, complex
- string
- tuple
- frozen set
- bytes

The following objects are **mutable**:

- list
- dict
- set
- byte array

Mutable: Changable after creation

Immutable: Unchangable after creation

Lists are mutable because they can add new elements in place
A string is immutable because changing a string will create a new string to bind with
Changing mutable objects is cheap
Changing immutable objects requires a lot of processing

Ex. concatenate a string requires a new string to be created

Each new assignment assigns a variable name to an object. These will point to the same object

Ex. a = blank
B = a

If blank is mutable, changing a or b will change it for the other
If it is immutable a copy of the variable will be made for b and they will be separate

Pass a mutable object to a function will really be passing a reference to the object, and therefore manipulating it within the function will seem as though the object is outside of the scope of the object

There are some situations where an immutable object can be changed

Ex. a tuple is immutable, but contains vars which may be mutable

Changing a mutable var within a tuple is legal

Takeaway: It is the **binding** which is mutable or immutable

Heap vs Stack

Everything in python is an object, and it sits on the heap, including int and float. The only things on the stack are the names of the variables, which point to their contents on the heap. C++, in contrast, can put int, float, and class (struct) on the stack

Scope

Scope can be visually represented by tabs

Modules

Module is a file which defines functions and may be imported in a project
An import will effectively import the name of a module, and the table of vars/functions
That way a function can be referenced via module.func()
If you change a module, you must create a new interpreter session to refresh the import
A .pyc file is a precompiled binary file

Example import
Users of the package can import individual modules from the package, for example:

import sound.effects.echo

This loads the submodule **sound.effects.echo**. It must be referenced with its full name.

sound.effects.echo.echofilter(input, output, delay=0.7, atten=4)

An alternative way of importing the submodule is:

from sound.effects import echo

This also loads the submodule **echo**, and makes it available without its package prefix, so it can be used as follows:

echo.echofilter(input, output, delay=0.7, atten=4)

Yet another variation is to import the desired function or variable directly:

from sound.effects.echo import echofilter

Again, this loads the submodule **echo**, but this makes its function **echofilter()** directly available:

echofilter(input, output, delay=0.7, atten=4)

Note that when using from package import item, the item can be either a submodule (or subpackage) of the package, or some other name defined in the package, like a function, class or variable. The import statement first tests whether the item is defined in the package; if not, it assumes it is a module and attempts to load it. If it fails to find it, an **ImportError** exception is raised.
Contrarily, when using syntax like import item.subitem.subsubitem, each item except for the last must be a package; the last item can be a module or a package but can't be a class or function or variable defined in the previous item.

from module import * is ill advised

This will fill your program with tables for every function specified in the module

Also, you may have function names within your own code which are the same in the module, resulting in conflicts

Functions

Functions are done with the format

def funcname():

Code

#Be aware of the colon and the indentation, python is format aware

Lists

A list is a datastructure that mirrors a vector in c++

Is considered mutable

Items can be added to a list with .append(var)

List[index1:index2] returns partial list from the given indices

References the items stored at index1 through index2 not including index2

.index(element) will return the index of said element

.insert(index, element) will insert an element at said index

.sort() will organize numerically or alphabetically

Sort will alter the original array, not return a new one

.remove() remove it

.pop() will remove an item on the list and return it to you

List comprehension uses if and for within creation

evens_to_50 = [i for i in range(51) if i % 2 == 0]

range(3)

Produces list [0, 1, 2]

Note: Parentheses refer to a tuple, brackets refer to a list

Tuples

Same ordered set up as a list

Immutable, and therefore no internal methods

Are faster than lists, if you have a set of constants that are only being iterated through use a set of tuples

Use parentheses in instantiation, then use bracket notation for accessing

For loops

Syntax: for variable in list_name:

```
for item in list:
    print item

for i in range(len(list)):
    print list[i]
for index, item in enumerate(choices):
    print index+1, item
list_a = [3, 9, 17, 15, 19]
list_b = [2, 4, 8, 10, 30, 40, 50, 60, 70, 80, 90]

for a, b in zip(list_a, list_b):
    # Add your code here!
    if a>b:
        print a
    else:
        print b
```

Dictionaries

Unordered list of keys with pair

d = {'key1' : 1, 'key2' : 2, 'key3' : 3}

D[key1] = number

Is considered mutable

You can put lists within dictionaries

Del key will remove key

.keys() returns list of keys

.values() returns a list of values

.items() returns tuples (both)

Classes

class Triangle(object): where object is the parent class

Since 3.0, super() will call the parent version of the function

No need to pass self to parent version

super().__init__(first, last, age)

Useful with inheritance

Other

The , character after our print statement means that our next print statement keeps printing on the same line.

raw_input("Your guess: ")

Will wait for user input

Seems like cin or scanf without flushing or complication

Lambda functions

Functional programming, passing functions as if they are variables

Anonymous functions that are not explicitly named, used for quick functionality

If you will used the code many times it is better to use def

filter(lambda x: x % 3 == 0) is the same as

```
def by_three(x):
   return x % 3 == 0
```

A filter lambda is applied to limit the input from a list to a function or operation

Map

Apply a function to every item of iterable and return a list of the results

Set

Another data structure that is immutable and can be used for set theory operations like intersection, union, etc