CS 142
Python Memory Model

Rhodes College

- Write Date Class
- List objects

Garbage Collection

- Def: deallocating memory for objects when they can no longer be accessed
  - Garbage collector removes all local variables in a function once the function call has been completed.
  - As the programmer, you can explicitly delete objects by removing the name from the namespace

    del <object_name>

    For the classes you write, you can write destructors (opposite of a constructor)

    * these become more important in C++, so we'll talk more about them then.

Garbage Collection

```
>>> l = [1, 2, 3, 4, 5, 6]
>>> l
[1, 2, 3, 4, 5, 6]
>>> locals()
{'__builtins__': <module 'builtins' (built-in)>, '__name__': '__main__',
'l': [1, 2, 3, 4, 5, 6], '__loader__': <class '_frozen_importlib.Builtin
Importer'>, '__doc__': None, '__spec__': None, '__package__': None}
>>> del l
>>> locals()
{'__builtins__': <module 'builtins' (built-in)>, '__name__': '__main__',
'__loader__': <class '_frozen_importlib.BuiltinImporter'>, '__doc__': No
ne, '__spec__': None, '__package__': None}
>>> l
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    l
NameError: name 'l' is not defined
```

Aliasing

- Multiple names refer to the same object

```
>>> lst1 = [1, 2, 3]
>>> lst2 = lst1
>>> lst2
[1, 2, 3]
>>> lst2.append(4)
>>> lst2
[1, 2, 3, 4]
>>> lst1
[1, 2, 3, 4]
>>> id(lst1)
44953808
>>> id(lst2)
44953808
```

```
>>> c = 7
>>> a = c
>>> id(a)
504610472
>>> id(c)
504610472
>>> a = 8
>>> c
7
>>> id(a)
504610488
>>> id(c)
504610472
```

Reassigns a new value to a here; uses different memory address

Shallow Copies vs. Deep Copies

- Shallow copy – underlying objects point to the same memory address, but the memory address of object name is different
- Deep copy – completely separate copy creates both references and new data objects (if necessary) at all levels

```
>>> import copy
>>> b = [3, 4, [5, 6], 8]
>>> c = b
>>> d = copy.copy(b)     #creates a shallow copy
>>> e = copy.deepcopy(b) #creates a deep copy
>>> b
[3, 4, [5, 6], 8]
>>> c
[3, 4, [5, 6], 8]
>>> d
[3, 4, [5, 6], 8]
>>> e
[3, 4, [5, 6], 8]
>>> b is c, b == c
(True, True)
>>> b is d, b == d
(False, True)
>>> b is e, b == e
(False, True)
>>> id(b), id(c), id(d), id(e)
(43945416, 43945416, 44376352, 747400)
```


Shallow Copies vs. Deep Copies

```
>>> c[0] = 1
>>> b, c, d, e
([1, 4, [5, 6], 8], [1, 4, [5, 6], 8], [3, 4, [5, 6], 8], [3, 4, [5, 6], 8])
>>> d[0] = 2
>>> b, c, d, e
([1, 4, [5, 6], 8], [1, 4, [5, 6], 8], [2, 4, [5, 6], 8], [3, 4, [5, 6], 8])
>>> e[0] = 9
>>> b, c, d, e
([1, 4, [5, 6], 8], [1, 4, [5, 6], 8], [2, 4, [5, 6], 8], [9, 4, [5, 6], 8])
>>> d[2][0] = 2
>>> b, c, d, e
([1, 4, [2, 6], 8], [1, 4, [2, 6], 8], [2, 4, [2, 6], 8], [9, 4, [5, 6], 8])
>>> e[2][1] = 3
>>> b, c, d, e
([1, 4, [2, 6], 8], [1, 4, [2, 6], 8], [2, 4, [2, 6], 8], [9, 4, [5, 3], 8])
```


Passing Parameters

```
def foo(e, f, g):
    print(id(e), id(f), id(g))
    e += 3
    f.append(9)
    g = [4, 8]
    print(id(e), id(f), id(g))
    print(e, f, g)

def main():
    b = 0
    c = [1, 2, 3]
    d = [5, 6, 7]
    print(b, c, d)
    print(id(b), id(c), id(d))
    foo(b, c, d)
    print(id(b), id(c), id(d))
    print(b, c, d)

main()
```

Output:

```
0 [1, 2, 3] [5, 6, 7]
504610360 43353192 4989336
504610360 43353192 4989336
504610408 43353192 43379128
3 [1, 2, 3, 9] [4, 8]
504610360 43353192 4989336
0 [1, 2, 3, 9] [5, 6, 7]
```