


Copying Mutables & More Functions


CSE 231



Mutable & Immutable

- Immutable
 - strings
 - tuples
 - basic types (int, float, etc...)
- Mutable
 - lists
 - sets
 - dictionaries

Mutable: capable of change or of being changed; from Latin *mutare* to change



Michigan State University
CSE 231, Fall 09 More Functions


Strings are immutable

```
>>> s = 'hi'
>>> s[0] = 'N'
```

Traceback (most recent call last):
File "<pyshell#3>", line 1, in <module>
s[0]='N'

TypeError: 'str' object does not support item assignment

```
>>> s.replace('h','N')
'Ni'
>>> s
'hi'
>>> new_s = s.replace('h','N')
>>> new_s
'Ni'
```



Michigan State University
CSE 231, Fall 09 More Functions


Tuples are immutable

```
>>> t = (1, 'a', [2,'b'])
>>> t
(1, 'a', [2, 'b'])
>>> print t[2] # prints [2,'b']
>>> t[2] = 2.5
```

Traceback (most recent call last):
File "<pyshell#25>", line 1, in <module>
t[2] = 2.5

TypeError: 'tuple' object does not support item assignment


```
>>> t[2][0]=7.5 # list within tuple remains mutable
>>> t
(1, 'a', [7.5, 'b'])
```



Michigan State University
CSE 231, Fall 09 More Functions

The is function

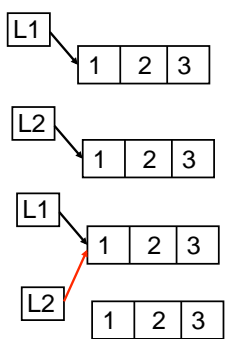

- Unlike ==, `is` measures whether two variables refer to exactly the same object (not just that the two objects have the same value)
- Only useful for mutables, its results are "implementation dependent" on immutables



Michigan State University
CSE 231, Fall 09 More Functions

Comparisons

```
>>> L1 = [1,2,3]
>>> L2 = [1,2,3]
>>> L1 == L2, L1 is L2
(True, False)
>>> L1 = L2
>>> L1 == L2, L1 is L2
(True, True)
```

Michigan State University
CSE 231, Fall 09 More Functions

Not a copy

```

>>> A = [1,2,3]
>>> B = A
>>> A == B, A is B
(True, True)
>>> B[2]=5
>>> A
[1, 2, 5]
    
```

Michigan State University
CSE 231, Fall 09
More Functions

Copy: shallow

```

>>> A = [1,2,3]
>>> B=A[:] or B=list(A)
>>> A==B, A is B
(True, False)
>>> B[2]=5
>>> A
[1, 2, 3]
>>> B
[1, 2, 5]
    
```

Michigan State University
CSE 231, Fall 09
More Functions

Copy: shallow

Var name is A
Var val is arrow

```

>>> A = [1,2,3]
>>> L = ['x',A,'y']
>>> L
['x', [1, 2, 3], 'y']
>>> M = L[:] or M=list(L)
>>> L == M, L is M
(True, False)
    
```

Michigan State University
CSE 231, Fall 09
More Functions

Copy: shallow

```

>>> M[0]= 'a'
>>> M
['a', [1, 2, 3], 'y']
>>> L
['x', [1, 2, 3], 'y']
>>> A[1]=55
>>> M
['a', [1, 55, 3], 'y']
>>> L
['x', [1, 55, 3], 'y']
    
```

Michigan State University
CSE 231, Fall 09
More Functions

Copy: deep

```

> import copy
> A = [1,2,3]
> L = ['x',A,'y']
> M = copy.deepcopy(L)
> L[1]==M[1], L[1] is M[1]
(True, False)
> A[1]=55
> M
['x', [1, 2, 3], 'y']
> L
['x', [1, 55, 3], 'y']
    
```

Note that the list embedded in M no longer has a name.

Michigan State University
CSE 231, Fall 09
More Functions

Copy for basic types

```

>>> x = 'abc'
>>> y = x
>>> y == x, x is y
(True, True)
>>> x = x.replace('b','z')
>>> y==x, x is y
(False, False)
>>> x,y
('azc', 'abc')
>>> x=x.replace('z','b')
>>> y==x, x is y
(True, False)
    
```

Michigan State University
CSE 231, Fall 09
More Functions

More on Functions



Functions return one thing

- Functions return one thing, but it can be a 'chunky' thing. For example, it can return a tuple
- Thus, multiple things can be returned by being packed into a tuple or other data structure



Michigan State University
CSE 231, Fall 09

More Functions

Functions return tuples

```
>>> def foo():
    a = 2
    b = 3
    return a,b

>>> T = foo()
>>> print T           # (2, 3)
>>> print foo()      # (2, 3)
>>> x,y = foo()
>>> print x           # 2
>>> print y           # 3
```



Michigan State University
CSE 231, Fall 09

More Functions

A fn where only mutables work

```
>>> def foo(a):
    a[1]='x'
    return a

>>> foo(2)           # Error
>>> foo('abc')      # Error
>>> foo([1,2,3])
[1, 'x', 3]
>>> foo((1,2,3))    # Error
```



Michigan State University
CSE 231, Fall 09

More Functions

assignment in a function

- if you assign a value in a function, that name becomes part of the local namespace of the function
- it can have some odd effects



Michigan State University
CSE 231, Fall 09

More Functions

Example

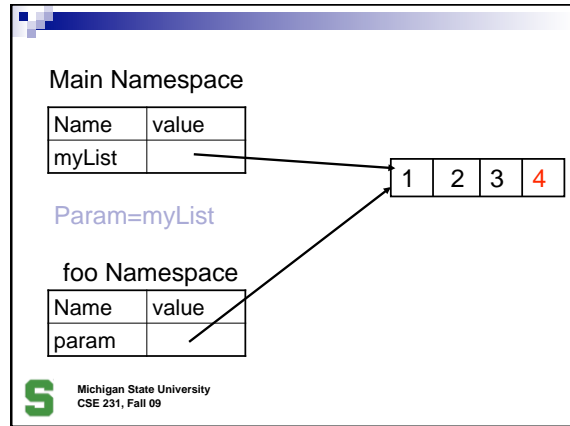
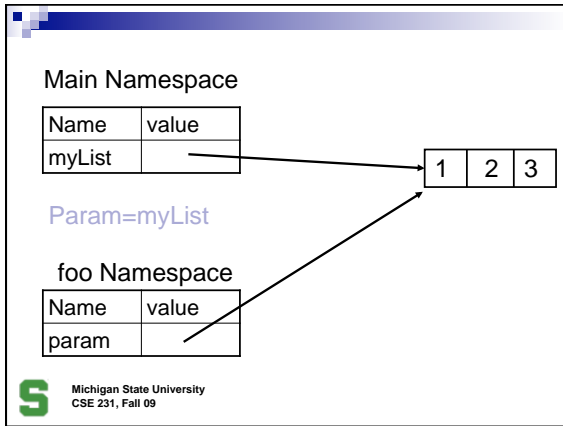
```
def myFun (param):
    param.append(4)
    return param

myList = [1,2,3]
newList = myFun(myList)
print myList,newList
```



Michigan State University
CSE 231, Fall 09

More Functions



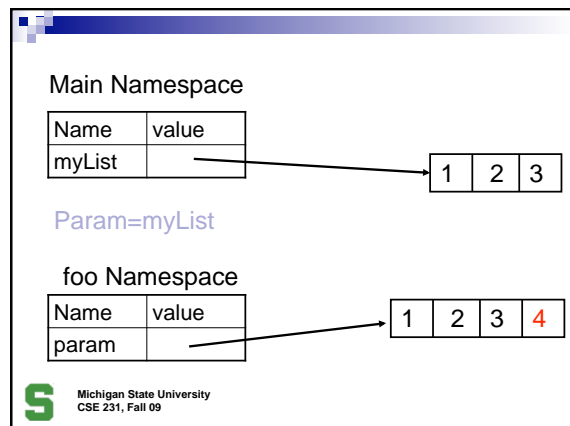
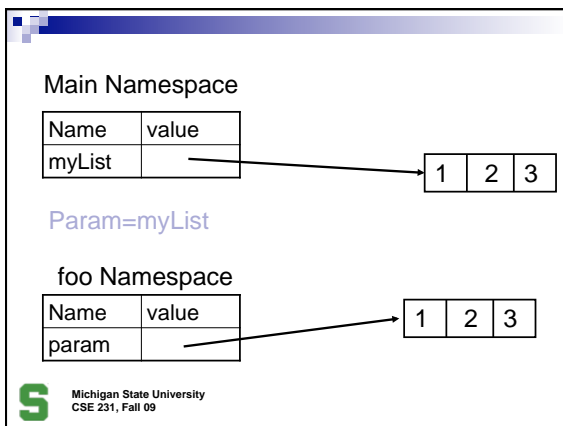
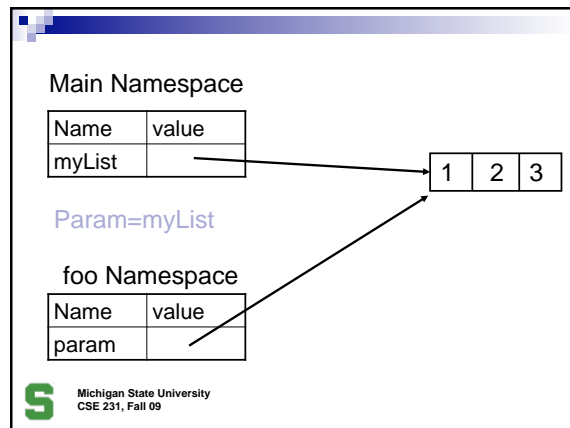
Example

```
def myFun (param):
    param=[1,2,3]
    param.append(4)
    return param

myList = [1,2,3]
newList = myFun(myList)
print myList,newList
```

Michigan State University
CSE 231, Fall 09

More Functions



Example

```
def myFun (param):
    param=param.append(4)
    return param
```

```
myList = [1,2,3]
newList = myFun(myList)
print myList,newList
```



Michigan State University
CSE 231, Fall 09

More Functions

Main Namespace

| Name | value |
|--------|-------|
| myList | |

Param=myList

foo Namespace

| Name | value |
|-------|-------|
| param | |



Michigan State University
CSE 231, Fall 09

Main Namespace

| Name | value |
|--------|-------|
| myList | |

Param=myList

foo Namespace

| Name | value |
|-------|-------|
| param | |



Michigan State University
CSE 231, Fall 09

Main Namespace

| Name | value |
|--------|-------|
| myList | |

Param=myList

foo Namespace

| Name | value |
|-------|-------|
| param | None |



Michigan State University
CSE 231, Fall 09

assignment to a local

- assignment creates a local variable
- changes to a local variable affects only the local context, even if it is a parameter and mutable
- If a variable is assigned locally, cannot reference it before this assignment, even if it exists in main as well



Michigan State University
CSE 231, Fall 09

More Functions

Default and Named parameters

```
def box(hei ght=10, wi dth=10, depth=10,
        col or= "bl ue" ):
    ... do something ...
```

The parameter assignment means two things:

- if the caller does not provide a value, the default is the parameter assigned value
- you can get around the order of parameters by using the name



Michigan State University
CSE 231, Fall 09

More Functions

Defaults

```
def box (height=10,width=10,length=10):
    print height,width,length

box()    # prints 10 10 10
```



Michigan State University
CSE 231, Fall 09

More Functions

Named parameter

```
def box (height=10,width=10,length=10):
    print height,width,length

box(length=25,height=25)
    # prints 25 10 25

box(15,15,15)    # prints 15 15 15
```



Michigan State University
CSE 231, Fall 09

More Functions

Name use works in general case

- def foo(a,b):
 - print a,b
- foo(1,2) # prints 1 2
- foo(b=1,a=2) # prints 2 1



Michigan State University
CSE 231, Fall 09

More Functions

Default args and mutables

- One of the problem with default args occurs with mutables. This is because:
 - the default value is created **once**, when the function is defined, and stored in the function name space
 - a mutable can change that value of that default



Michigan State University
CSE 231, Fall 09

More Functions

weird

```
def fn1 (arg1=[], arg2=27):
    arg1.append(arg2)
    return arg1

myList = [1,2,3]
print fn1(myList,4)    # [1, 2, 3, 4]
print fn1(myList)# [1, 2, 3, 4, 27]
print fn1()            # [27]
print fn1()            # [27, 27]
```



Michigan State University
CSE 231, Fall 09

More Functions

print fn1() arg1 is either assigned to the passed arg or to the function default for the arg

fn1 Namespace

| Name | Value |
|------|-------|
| arg1 | |
| arg2 | |

Diagram illustrating the fn1 namespace. The table shows 'arg1' and 'arg2' as names with empty value cells. Arrows point from the 'arg1' cell to an empty box, and from the 'arg2' cell to the value '27'.



Michigan State University
CSE 231, Fall 09

More Functions

`print fn1()` Now the function default, a mutable, is updated and will remain so for the next call

fn1 Namespace

| Name | Value |
|------|-------|
| arg1 | 27 |
| arg2 | 27 |

Michigan State University
CSE 231, Fall 09

Docstrings

Michigan State University
CSE 231, Fall 09

Docstring

- If the first item after the def is a string, then that string is specially stored as the docString of the function
- This string describes the function and is what is shown if you do a help on a function
- Usually triple quoted since it is multiline

Michigan State University
CSE 231, Fall 09

Docstring

```
def listMean(myList):
    '''Takes a list of integers, returns
    the average of the list.'''
    return float(sum(myList))/len(myList)
```

Michigan State University
CSE 231, Fall 09

Can ask for docstring

- Every object (function, whatever) can have a docstring
- `listMean.__doc__`
 - `'Takes a list of integers, returns the average of the list.'`
- `.__attribute__` are reserved elements in Python

Michigan State University
CSE 231, Fall 09


Arbitrary arguments

- it is also possible to pass an arbitrary number of arguments to a function
- the function simply collects all the arguments (no matter how few or many) into a tuple to be processed by the function
- tuple parameter preceded by a * (which is not part of the param name, its part of the language)
- positional arguments only, no named params

Michigan State University
CSE 231, Fall 09


example

```
def aFunc(fixedParam,*tupleParam):
    print `fixed =`,fixedParam
    print `tuple=`,tupleParam
aFunc(1,2,3,4)
prints      fixed=1
            tuple=(2,3,4)
aFunc(1)
prints      fixed=1
            tuple=()
aFunc(fixedParam=4)
prints      fixed=1
            tuple=()
aFunc(tupleParam=(1,2,3),fixedParam=1)
error
```

 Michigan State University
CSE 231, Fall 09

More Functions

Worksheet 9

 More Functions