Python 101

Mutability and Aliasing

Announcements

quiz: `quiz07` due on Thurs 10/10

quiz: `quiz06` reopened and due on 10/10.

lab: `lab04` on 11/10

hw: `hw04` due Wed 16/10 (delayed..)

exam: `exam01` on `12/10 6-7 pm`

Correction - Lec05 Func Slide 16

M3

```
lettE = 0
lettCap = 0
lengthS = len(tryI)
for letter in tryI:
    if letter in 'abcdefghijklmnopqrstuvwxyz
                  ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789':
        lettE += 1
if not (tryI.isupper() or tryI.islower()
   or tryI.isdigit() or tryI.isalpha()): ##added
        lettCap = 1
if lettE == lengthS and lengthS >= 8
                                    and lettCap > 0:
    print("Password OK")
else:
    print( 'Invalid password' )
```

OK

Correction - Lec05 Func Slide 18

Sort small to big (ascending order)
- `x.sort()`

Sort big to small (descending order)
- `x.sort(reverse=True)`

not `x.reverse()`

Correction - Lec03 Bool Slide 19

Correction - Lec03 Bool Slide 19

Order (highest priority listed first):

`<, <=, >, >=` same order in this line

`==, !=` same order in this line

`not`

`and`

`or`

Correction - Lec03 Bool Slide 19

Order (highest priority listed first):

`<, <=, >, >=` same order in this line

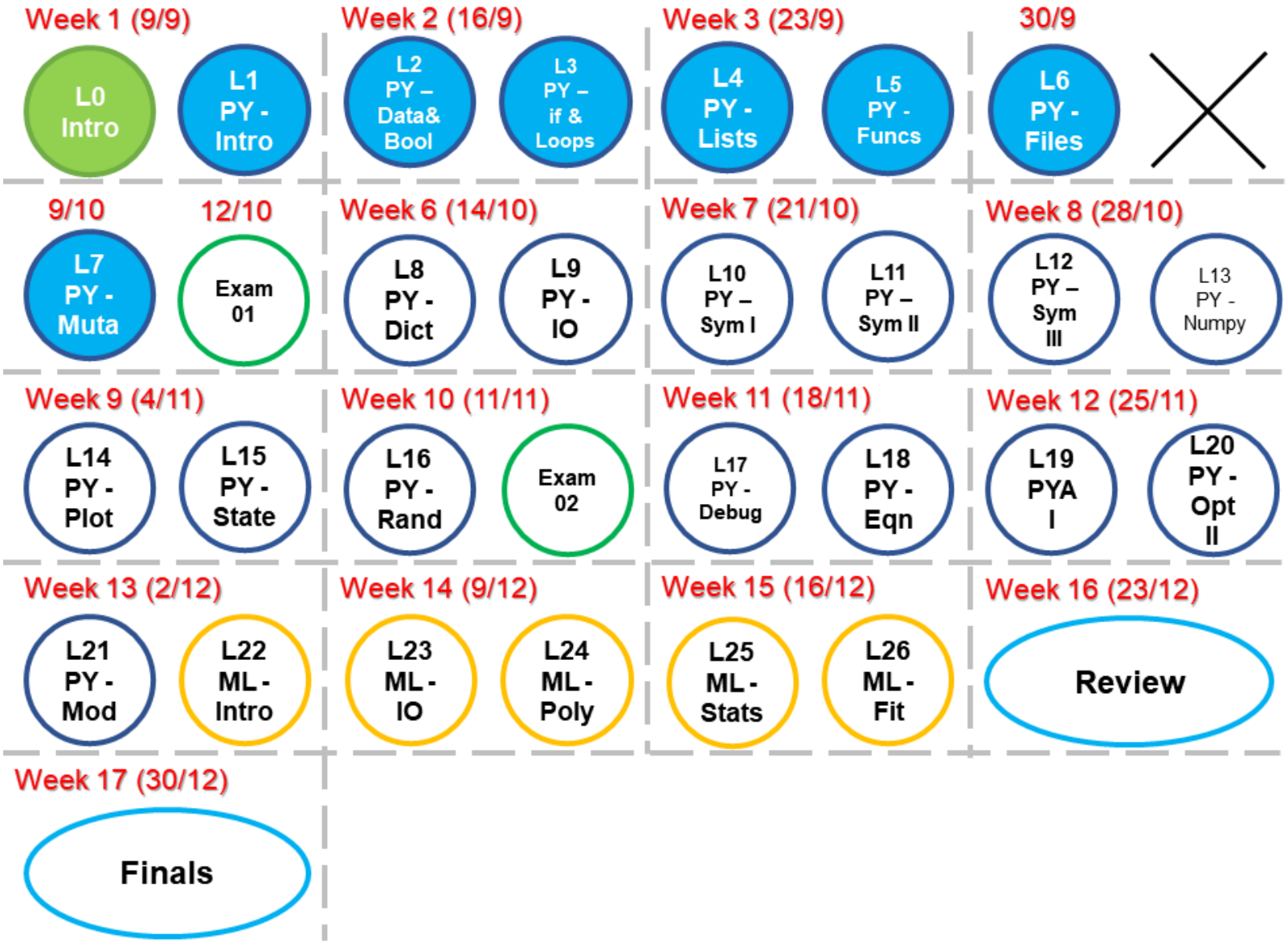
`==, !=` same order in this line

`not`

`and`

`or`

Roadmap


Week 1 (9/9)
L0 Intro
L1 PY - Intro
Week 2 (16/9)
L2 PY – Data& Bool
L3 PY – if & Loops
Week 3 (23/9)
L4 PY - Lists
L5 PY - Funcs
30/9
L6 PY - Files
9/10
L7 PY - Muta
12/10
Exam 01
Week 6 (14/10)
L8 PY - Dict
L9 PY - IO
Week 7 (21/10)
L10 PY – Sym I
L11 PY – Sym II
Week 8 (28/10)
L12 PY – Sym III
L13 PY - Numpy
Week 9 (4/11)
L14 PY - Plot
L15 PY - State
Week 10 (11/11)
L16 PY - Rand
Exam 02
Week 11 (18/11)
L17 PY - Debug
L18 PY - Eqn
Week 12 (25/11)
L19 PYA I
L20 PY - Opt II
Week 13 (2/12)
L21 PY - Mod
L22 ML - Intro
Week 14 (9/12)
L23 ML - IO
L24 ML - Poly
Week 15 (16/12)
L25 ML - Stats
L26 ML - Fit
Week 16 (23/12)
Review
Week 17 (30/12)
Finals

Objectives

A. Understand what mutability is and why some variables can be changed, other not.
B. Understand when and why aliasing occurs.
C. Identify `tuples`.

File Recap

Question

```
myfile = open('words.txt')

myfileasstring = myfile.read()
myfileaslist = myfile.readlines()

myfile.close()
```

What is the final value of `myfileaslist`?

A `[ ]`

B The contents of the file as a list of strings.

C `None`

Question

```
myfile = open('words.txt')

myfileasstring = myfile.read()
myfileaslist = myfile.readlines()

myfile.close()
```

What is the final value of `myfileaslist`?

A `[ ]` ⋆Why?

B The contents of the file as a list of strings.

C `None`

string.split example

`split` a `string` returns a `list`.

Takes a single string argument, the *delimiter*.

```
name = 'Catherine?Bourne?Angel'
names1 = name.split( ' ' )
names2 = name.split( '?' )
```

Ans:

string.split example

`split` a `string` returns a `list`.

Takes a single string argument, the *delimiter*.

```
name = 'Catherine?Bourne?Angel'
names1 = name.split( ' ' )
names2 = name.split( '?' )
```

Ans:
names1 = 'Catherine?Bourne?Angel'
names2 = ['Catherine', 'Bourne', 'Angel']

string.join example

```
names = [ "Nancy", "Chris",
          "Tiffany", "Tony" ]

' likes '.join( names )      # note the odd syntax!
                      # join is a STRING method
```

Ans:

string.join example

```
names = [ "Nancy", "Chris",
          "Tiffany", "Tony" ]

' likes '.join( names )      # note the odd syntax!
                       # join is a STRING method
```

Ans: `'Nancy likes Chris likes Tiffany likes Tony'`

Mutability

Example

```
x = 1
y = x
y = 2
# what is x and type()?
```

x = 1 and `int`

Example

```
x = 1
y = x
y = 2
# what is x and type()?
```

x = 1 and `int`

```
x = [ 1,2,3 ]
y = x
y[0] = 6
# what is x and type()?
```

Example

```
x = 1
y = x
y = 2
# what is x and type()?
```

x = 1 and `int`

```
x = [ 1,2,3 ]
y = x
y[0] = 6
# what is x and type()?
```

x = [6,2,3] and `list`

Mutability

We distinguish *mutability* and *immutability*.

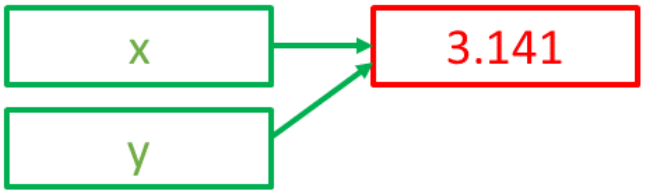
The distinction arises from the storage in memory.

Mutability

Immutability describes objects with values that are not changeable in memory, often when values are copies.

Basic Python data types - `str`, `int`, `float`, `complex` are *immutable*

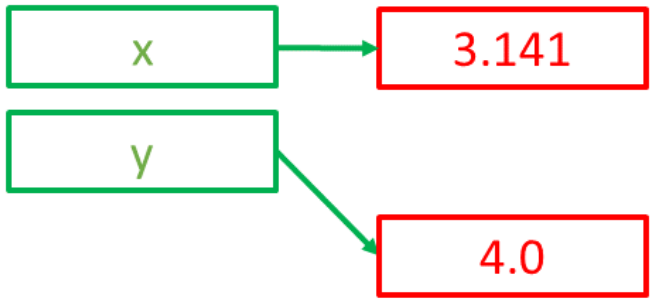
```
x = 3.141

y = x
```



Mutability

Immutability describes objects with values that are not changeable in memory, often when values are copies.

```
x = 3.141

y = x

y = 4.0
```

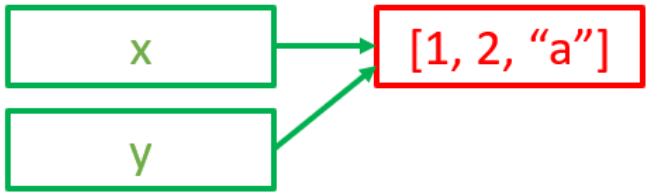


Mutability & immutability

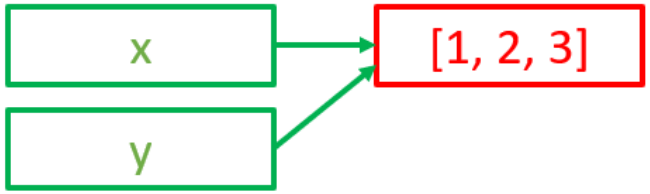
Mutability describes values that are changeable in memory, often when values share the same location.

python types `list` and `dict` are mutable

```
x = [1, 2, "a"]

y = x
```




```
x = [1, 2, "a"]

y = x

y[2] = 3
```



Aliasing

Aliasing occurs when one memory location has two names.

Aliasing causes mutable types to behave unexpectedly!

Aliasing

```
x = [ 1, 2, 3, 4 ]
y = x
x[ -1 ] = 2
```

Example

```
a = [ 'a', 'b', 'c', 'd' ]
b = a
b[3] = '*'
```

What is the final value of `a`?

A `[ 'a', 'b', '*', 'd' ]`

B `[ 'a', 'b', 'c', '*' ]`

C `[ 'a', 'b', 'c', 'd' ]`

D None of the above.

Example

```
a = [ 'a', 'b', 'c', 'd' ]
b = a
b[3] = '*'
```

What is the final value of `a`?

A `[ 'a', 'b', '*', 'd' ]`

B `[ 'a', 'b', 'c', '*' ]` ⋆

C `[ 'a', 'b', 'c', 'd' ]`

D None of the above.

String statement

Strings are *immutable* (we can't change contents without creating a new string):

(

```
s = ”good advise”
s[9] = ’c’
# Error!!!
```

String statement

Strings are *immutable* (we can't change contents without creating a new string):

(

```
s = "good advise"
s[9] = 'c'
# Error!!!

s = s[:9] + 'c' + s[10:]     # ok
```

Tuples

Tuples

Another python data type!

The immutable analogue of a `list` is a `tuple`.

We form a tuple by using parentheses `()` instead of square brackets `[]`.

Where can I use tuples?

`tuple`s can be used to format multiple values for `print`.

```
'%i %i %i' % (1,2,3)
```

Example

```
s = ???
x = 10
y = 'Hello'
z = 3.14
print(s % x,y,z)
```

What should replace the `???`?

A `'%i %f %s'`

B `'%f %s %i'`

C `'%i %s %f'`

D None of the above.

Example

```
s = ???
x = 10
y = 'Hello'
z = 3.14
print(s % x,y,z)
```

What should replace the `???`?

A `'%i %f %s'`

B `'%f %s %i'`

C `'%i %s %f'` ⋆

D None of the above.

Where can I use tuples?

`tuple`s can return *multiple values* from a function.

(You've seen this a couple of times before.)

```
def fun():
    return 'hi', 3, 'lo'

retVal = fun()
```

Ans:

Where can I use tuples?

`tuple`s can return *multiple values* from a function.

(You've seen this a couple of times before.)

```
def fun():
    return 'hi', 3, 'lo'

retVal = fun()
```

Ans: retVal = ('hi', 3, 'lo')

Where can I use tuples?

`tuple`s can also be used for *multiple assignments* at once.

```
one,pi,hello = ( 1,3.14,'Hi' )
```

Where can I use tuples?

`tuple`s can also be used for *multiple assignments* at once.

```
one,pi,hello = ( 1,3.14,'Hi' )
```

of course you can also do:

```
one,pi,hello = 1, 3.14, 'Hi'
```

Where can I use tuples?

`( 1.0 ) is a float not a tuple`

How do we make a one-element `tuple`?

Where can I use tuples?

`( 1.0 ) is a float not a tuple`

How do we make a one-element `tuple`?

```
( 1.0, )
```

Container Methods for list

This explains a bit about container methods for `list`!

Container Methods for list

This explains a bit about container methods for `list`!

`sort` and `append` modify the `list` itself.

Warning!

This explains why `sort` and `append` `return None`!

```
x = [ 4,1,2,3 ]
x.sort()  #This is the right way to sort a list.
print(x)
```

Container Methods - list

`sort`, `reverse`, and `append` modify the `list` itself.

Warning!

This explains why `sort` and `append` `return None`!

```
x = [ 4,1,2,3 ]
x = x.sort()     # MANY of you will do this.
print(x)
```

Example

```
y = [ 3,2,1 ]
x = y.append( 5 )
y[-1] = 3
```

What is the final value of `x`?

A `[ 3, 2, 1, 3 ]`

B `[ 3, 2, 1, 5 ]`

C `[ 3, 2, 1 ]`

D `None`

Example

```
y = [ 3,2,1 ]
x = y.append( 5 )
y[-1] = 3
```

What is the final value of `x`?

A `[ 3, 2, 1, 3 ]`

B `[ 3, 2, 1, 5 ]`

C `[ 3, 2, 1 ]`

D `None` ⋆

Container Methods

`index` returns the index of the first occurrence of a value.

`count` returns how many times a value occurs.

`in` returns membership in a container.

`*` *repeats* a container.

`+` *extends* a container.

`max(x)`, `min(x)`, `len(x)`, etc.

Mutable Arguments

Exercise: mutability

```
x = [ 3,2,1 ]
y = x
y.sort()
x.append( 0 )
```

What is the final value of `x`?

A `[ 3,2,1 ]`

B `[ 1,2,3 ]`

C `[ 1,2,3,0 ]`

D `[ 0,1,2,3 ]`

Exercise: mutability

```
x = [ 3,2,1 ]
y = x
y.sort()
x.append( 0 )
```

What is the final value of `x`?

A `[ 3,2,1 ]`

B `[ 1,2,3 ]`

C `[ 1,2,3,0 ]` ⋆

D `[ 0,1,2,3 ]`

Mutable arguments

Mutability causes `list`s to work differently in functions.

`list`s used as arguments *can be changed* by the function.

This is very useful and tricky!

```
def appender( q ):
    q.append( 3 )

aList = [ ]
for i in range( 3 ):
    appender( aList )
print( aList )
```

Where does `aList` live? What is the value of `aList`?
Ans:

Mutable arguments

Mutability causes `list`s to work differently in functions.

`list`s used as arguments *can be changed* by the function.

This is very useful and tricky!

```
def appender( q ):
    q.append( 3 )

aList = [ ]
for i in range( 3 ):
    appender( aList )
print( aList )
```

Where does `aList` live? What is the value of `aList`?
Ans:
`aList` lives outside the function but can be changed by the function if passed into the function.

```
[3, 3, 3]
```

Mutable arguments for FILE ex1

```
def readfile( fname,a ):
    myfile = open( fname,'r' )
    for line in myfile.readlines():
        a.append(line)
    myfile.close()

all_lines = []
readfile( 'file1.txt',all_lines )
readfile( 'file2.txt',all_lines )
```

Mutable arguments for FILE ex2

```
def readfile( fname,a ):
    myfile = open( fname,'r' )
    for line in myfile.readlines():
        a.append(line)
    myfile.close()

all_lines = []
files = [ 'file1.txt','file2.txt','file3.txt' ]
for f in files:
    readfile( f,all_lines )
```

Copying mutable values

What if we *want* a copy of a `list` (not an alias)?

Copying mutable values

What if we *want* a copy of a `list` (not an alias)?

Slice everything!

```
x = [ 3,2,1 ]
y = x[ : ]
y.sort()
print( x )
print( y )
```

Copying mutable values

What if we *want* a copy of a `list` (not an alias)?

Slice everything!

```
x = [ 3,2,1 ]
y = x[ : ]
y.sort()
print( x )
print( y )
```

Ans:

x = [3, 2, 1]

y = [1, 2, 3]

Copying mutable values

```
x = [ 1,2,3 ]
y = x[ : ]
y.append( 4 )
print( x == y )
```

Copying mutable values

```
x = [ 1,2,3 ]
y = x[ : ]
y.append( 4 )
print( x == y )
```

Ans: False

Test mutable? is tests identity

```
a = [ 1,2,3 ]
b = a
c = a[ : ]

b is a
c is a
```

Test mutable? is tests identity

```
a = [ 1,2,3 ]
b = a
c = a[ : ]

b is a
c is a
```

Ans:
True
False

Test mutable? id tells identity

```
a = [ 1,2,3 ]
b = a
c = a[ : ]

id(a)
id(b)
id(c)
```

Test mutable? id tells identity

```
a = [ 1,2,3 ]
b = a
c = a[ : ]

id(a)
id(b)
id(c)
```

Ans:
2593298607112
2593298607112
2593298589640

Summary

Summary

A. `list` and `dict` are mutable and aliasing can occur
B. `int, float, str, tuple, complex` are not
C. `list` passed into function can be changed by function
D. `is` and `id`