Python Basics

Mutability, container methods

CS101 Lecture #9

2016-10-24

`For` loops

Examples

```
for i in range(10):
    print(i**2)
```

Examples

```
for i in range(10):
    print(i**2)

for i in range(2,10):
    print(i**2)
```

Examples

```
for i in range(10):
    print(i**2)

for i in range(2,10):
    print(i**2)

for i in range(2,10,3):     # arithmetic sequence from 2 to
                            # 10 (exclusive) w/ increment 3
    print(i**2)
```

Mutability and Aliasing

Examples

```
x = 1
y = x
y = 2

#What is the value of x?
```

Examples

```
x = 1
y = x
y = 2

#What is the value of x?

x = ‘Hello’
y = x
y = y + ‘ world!’

#What is the value of x?
```

Examples

```
x = 1
y = x
y = 2

#What is the value of x?

x = 'Hello'
y = x
y = y + ' world!'

#What is the value of x?

x = [1,2,3]
y = x
y[0] = 0

#What is the value of x?
```

Mutability

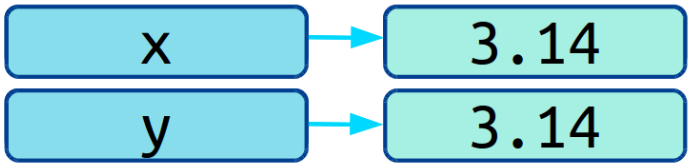
- Recall the distinction of mutability (of list) and immutability (of string)

- The distinction arises from the storage in memory

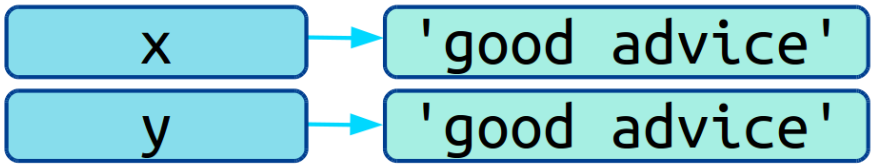
Mutability

- *Immutable types*: value of x is copied in memory and assigned to y *(copy by value)*

```
x = 3.14
y = x
```



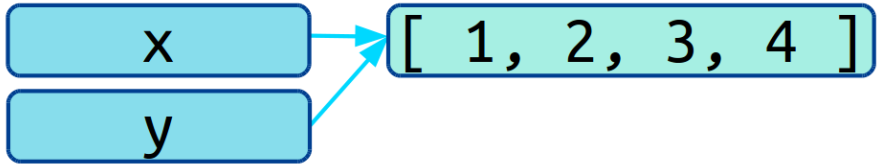

```
x = 'good advice'
y = x
```



Mutability

- Mutable type: the new variable y refers to the same memory location of the value that x refers to *(copy by reference, or shallow copy)*

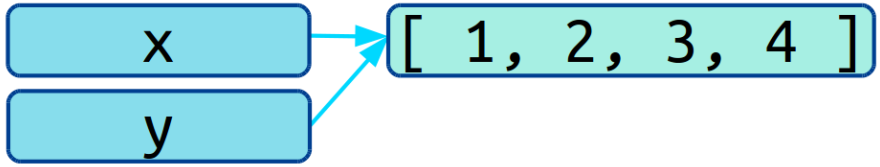
```
x = [ 1, 2, 3, 4 ]
y = x
```



Mutability

- Mutable type: the new variable y refers to the same memory location of the value that x refers to *(copy by reference, or shallow copy)*

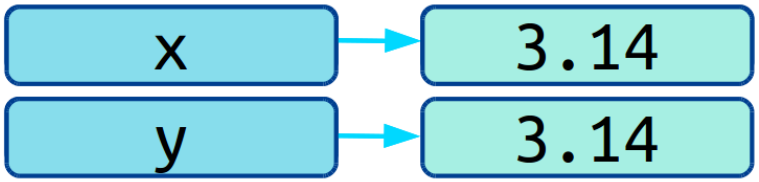
```
x = [ 1, 2, 3, 4 ]
y = x
```



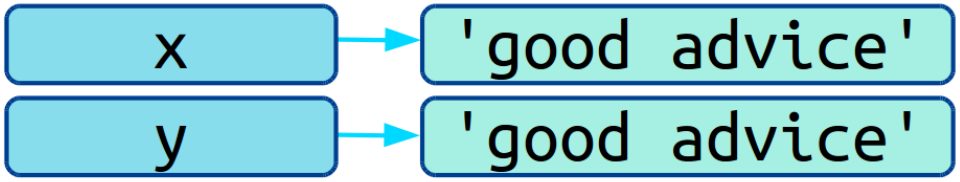
- Benefit: avoid large chunks of memory copy
- Drawback (?): aliasing

Mutability

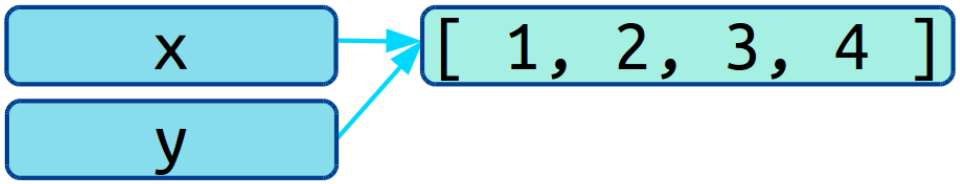
```
x = 3.14
y = x
```




```
x = 'good advice'
y = x
```




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



Mutability

- Python data types are either mutable or immutable
  - int, float, string, tuple
  - list, dictionary

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

#What is the value of y?
```

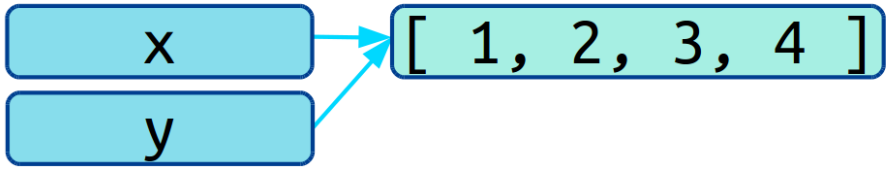
Aliasing

- Occurs in mutable types
- Changes in one variable causes value of another variable to change
- sometimes useful, sometimes cause trouble

Aliasing

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

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

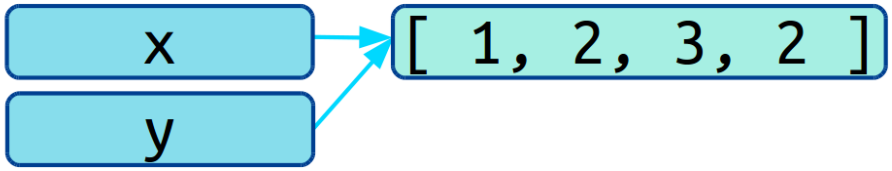


Aliasing

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

#What is the value of y?
```

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



Examples

```
x = ['a', 'b', 'c', 'd']   # a dictionary
y = x
y[2] = '*'

#What is the value of x?
 A ['a', 'b', 'c', 'd']
 B ['a', '*', 'c', 'd']
 C ['a', 'b', '*', 'd']
 D None of the above
```

Examples

```
x = {1:'a', 2:'b', 3:'c', 4:'d'}  # a dictionary
y = x
y[2] = '*'
```

What is the value of x?

A {1:'a', 2:'b', 3:'c', 4:'d'}
B {1:'a', 2:'b', 3:'*', 4:'d'}
C {1:'a', 2:'*', 3:'c', 4:'d'}
D None of the above

Tuples

- The immutable analogue of a `list`
- Use the parentheses () instead of the square brackets []
  - Parentheses can sometimes be skipped

```
x = [1,2,3,4]     # x is a list
y = (1,2,3,4)     # y is a tuple
y = 1,2,3,4       # equivalent to above line
```

Tuples

- Access elements by index

```
x = [1,2,3,4]     # x is a list
y = (1,2,3,4)     # y is a tuple
print(y[0])
print(y[-1])
```

Tuples

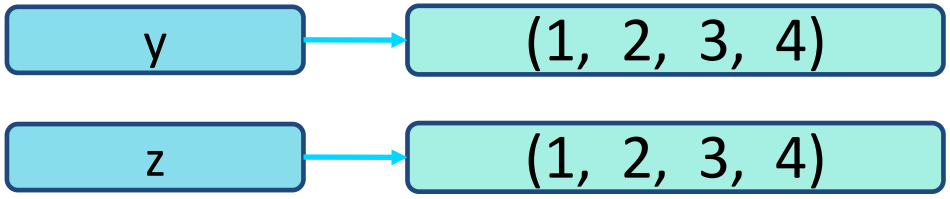
- Cannot change elements within a tuple (like a string)

```
x = [1,2,3,4]     # x is a list
y = (1,2,3,4)     # y is a tuple
y[0] = -1         # error!
```

Tuples

- Cannot change elements within a tuple (like a string)
- Copy by value

```
x = [1,2,3,4]     # x is a list
y = (1,2,3,4)     # y is a tuple
y[0] = -1         # error!
z = y
```



Where can we use tuples?

- Formatting multiple values in string format
  - Parentheses is required

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

Where can we use tuples?

- Formatting multiple values in string format
  - Parentheses is required

```
'%i %i %i'%(1,2,3)
'the avg. grade of HW%i is %.3f'%(4, 85.0)
```

Exercise

```
s = ???
x = 10
y = 'Hello'
Z = 3.14

print(s % x,y,z)
```

Which of the following are valid replacement of ???

A '%i %f %s'
B '%f %s %i'
C '%f %s %f'
D '%i %s %f'

Where can we use tuples?

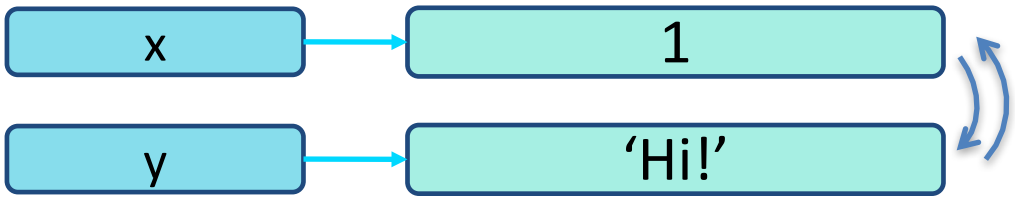
- Can also be used on the left hand side of an assignment operator
  - Lets us make multiple assignment at once
  - Parentheses is optional

```
One, pi, hello = (1, 3,14, 'Hi')
One, pi, hello =  1, 3,14, 'Hi'
```

Where can we use tuples?

- Can also be used on the left hand side of an assignment operator
  - Lets us make multiple assignment at once
  - Parentheses is optional

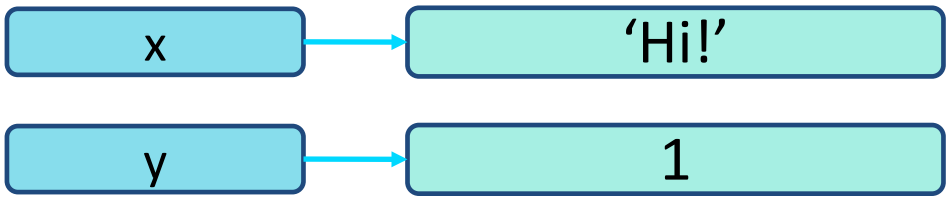
```
x,y = y,x          # exchange values of two variables
                   # parentheses is skipped
```



Where can we use tuples?

- Can also be used on the left hand side of an assignment operator
  - Lets us make multiple assignment at once
  - Parentheses is optional

```
x,y = y,x          # exchange values of two variables
                   # parentheses is skipped
```



Where can we use tuples?

- Return *multiple values* from a function

```
def foo():
    return 'hi', 3, '3.14'

a,b,c = fun()
```

Where can we use tuples?

- Return *multiple values* from a function

```
def fun(x):
    y = x**2
    z = x**0.5
    return y, z
```

Where can we use tuples?

- Return *multiple values* from a function

```
def fun(x):
    y = x**2
    z = x**0.5
    return y, z

>>> a,b = fun(10)
```

Where can we use tuples?

- Return *multiple values* from a function

```
def fun(x):
    y = x**2
    z = x**0.5
    return y, z

>>> a,b = fun(10)
>>> fun(10)[0]
100
>>> fun(10)[1]
3.1622…
```

`List` and `string` Methods

`List` *methods*

- Because lists are mutable, we can change their content by calling its method functions

```
x = [4,1,2,3]
x.append(5)
x.reverse()
x.sort()
```

`List` *methods*

- Because lists are mutable, we can change their content by calling its method functions

```
x = [4,1,2,3]
x.append(5)
x.reverse()
x.sort()

x = [3.14, 1, 'hi']
x.sort()                          #error!
```

Example

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

What is the value of y?
 A [3,2,1]
 B [3,2,3]
 C [3,2,1,5]
 D [3,2,1,3]
 E None of the above
```

`List` *methods*

- Append, reverse, sort modify the list content itself
- Returns a `NoneType`!

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

More *`List` methods*

- `index` returns the index of the first occurrence of a value in a list (`find` for str)
- `count` returns how many times a value occurs
- `in` returns membership (True or False) in a list
- `*` repeats a list
- `+` extends a list (also `extend`)
- `max`, `min`, `len`,…

`String` methods

- `split` splits a string by a delimiter
- Returns a list
- Takes a single argument, the delimiter

```
s = 'Mary,Watson,Holmes'
names = s.split(',')
print(names)
```

`String` methods

- `split` splits a string by a delimiter
- Returns a list
- Takes a single argument, the delimiter
  - Default in `' '`

```
s = 'Mary Watson Holmes'
names = s.split(' ')
print(names)

s = 'Mary Watson Holmes'
names = s.split()
print(names)
```

Example

```
x = 'A+B+C'
y = x.split('+')

What is the value of y?
 A ('A', 'B', 'C')
 B ['A', 'B', 'C']
 C ['A+B+C']
 D 'A+B+C'
 E None
```

Example

```
x = 'A+B+C'
y = x.split()

What is the value of y?
 A ('A', 'B', 'C')
 B ['A', 'B', 'C']
 C ['A+B+C']
 D 'A+B+C'
 E None
```

`String` methods

- `join` takes a list of string types and concatenate them together into a big string
- The reverse of `split`

```
names = ['Mary', 'Watson', 'Holmes']
#goal: 'Mary Watson Holmes'

' '.join(names)
```

`String` methods

- `join` takes a list of string types and concatenate them together into a big string
- The reverse of `split`

```
names = ['Mary', 'Watson', 'Holmes']
#goal: 'Mary Watson Holmes'

#also try:
''.join(names)
','join(names)
```

Example

```
a = ['X', 'A', 'G']
b = a
a.sort()
x = ','.join(b)
```

What is the value of x?

A ['A,G,X']
B ['X,A,G']
C 'X,A,G'
D 'A,G,X'

Example

```
a = ['X', 'A', 'G']
x = '#####'.join(a).split('#####')
```

What is the value of x?