CS101: Intro to Computing
Spring 2016

Lecture 10
Administrivia

• Homework 5 is due Friday
• Midterm 1 is February 29th at 7pm
  – Practice exam posted
  – NEXT MONDAY!!!
t=""
for c in s:
    if c not in "aeiou":
        t+=c

What is this program doing to string s?

a) Counting the vowels in s
b) Removing the vowels from s
c) Counting the consonants in s
d) Removing the consonants from s
```python
x=0
for i in [1,4950,99,100][0:-1]:
    x=i

What is the final value of x?

a) 0
b) 99
c) 100
d) 4950
```
LISTS
Lists

• Represents an ordered collection of *items* or *elements*.
  – It’s iterable.

• A *container* type
  – Contains other values of *any type*
  – **NOTE:** elements don’t have to be the same type
Lists

• We create an *list* by typing:
  1. an open square bracket [ 
  2. items of the list, separated by commas 
  3. a closing square bracket ]
Similarity to Strings

x=[10, 3.14, "Ride"]
print(x[1])
print(x[1:3])
print(len(x))
print(x*3)
for i in x:
    print i
Dissimilarity to Strings

• Strings are **immutable** (we can’t change the *contents* without **creating a new string**)

  \[
  s = "\text{Puraty Ring}"
  \]

  \[
  s[3] = "i" \quad \text{← NOT ALLOWED}
  \]

  \[
  s = s[0:3] + "i" + s[4:]
  \]
Immutable Assignment

\[ x = 3.14 \]

\[ y = x \]
Immutable Assignment

\[ s = \text{"Purity Ring"} \]
\[ t = s \]
Dissimilarity to Strings

• Lists are *mutable* (we *can* change the contents of a list)

```python
x=[4,1,2,3]
x[3]=-2  # item assignment
x.append(5)
del x[1]
x.sort()
```
Mutable Assignment

\[ a = [1, 2, 3, 4] \]

\[ b = a \]
Aliasing

- One memory location has two names.
- Only *mutable* types can be aliased.
- Aliasing causes mutable types to behave *very* differently.
Implications of Aliasing

\[ a = [1, 2, 3, 4] \]
\[ b = a \]
\[ b[-1] = 2 \]
Mutable arguments

• Mutability causes lists to work differently with functions.
• Lists used as arguments *can be modified by the function.*
• This is very useful.
```python
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]
```
The `sort` and `append` methods modify the list **itself**

This means they **RETURN NULL**

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

What is the final value of \( x \)?

a) \([3, 2, 1, 3]\)
b) \([3, 2, 1, 5]\)
c) \([3, 2, 1, 5, 3]\)
d) None
Mutable arguments

def fun(q):
    q.append(3)

a=[]
for i in range(3):
    fun(a)
print(a)
Example

def readfile(fname,a):
    for line in open(fname):
        a.append(line)

all_lines=[]
readfile("file1",all_lines)
readfile("file2",all_lines)
Example

def readfile(fname,a):
    for line in open(fname):
        a.append(line)

all_lines=[]
for f in open("filenames.txt"):  
    readfile(f,all_lines)
Copying Lists

• What if we want a fresh, independent copy of our list (i.e. NOT an alias?)

• **Slicing** creates a new list.
• Slice the **entire** list to create a copy.

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

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

• A **string** method that returns a **list**.
• Takes a single string argument.
  – Used as a delimiter

```python
name=“Ryan M. Cunningham”
m=name.split(“ ”)
print(m[-1])
```
x = "A+B+C"
y = x.split("+")

What is the value of x?
a) "ABC"
b) ["A","B","C"]
c) ["+","+","+"]
d) None
Join

• A string method that operates on a list.
• Returns a string of list elements joined together.

```python
names=["Ryan","Dave","Michael"]
'
'.join(names)
```
a = ["X", "A", "G"]
b = a[:]
a.sort()
x = "".join(b)

What is the value of x?

a) "XAG"
b) "X,A,G"
c) "A,G,X"
d) None
TUPLES
Tuple

• A tuple is an **immutable** sequence of any type.
  – An immutable version of a list.
• Literal: item in the tuple separated by commas (can add parentheses)
  \[ t=(1,3.14,\text{"Hi"}) \]
t=(1, 3.14, "Hi")
t[0:2]
t[-2]
len(t)
1 in t
t[2][1]
Why tuples?

• Less useful version of lists?
• No! They make our solutions more elegant!
• Allow us to group items together in our code.
Tuple in assignment

• A tuple can go on the *left side* of an assignment statement
• Allows us to make *multiple assignments* at once

```python
one, pi, hello = (1, 3.14, "Hi")
```

• Convenient for swapping values:

```python
x, y = y, x
```
Tuples as return values

• A tuple can be used in a return statement
• Allows us to *return multiple values* at once

```python
def fun():
    return (1, 2, 3)
```

• When calling, can use tuple assignment

```python
a, b, c = fun()
```
String formatting with tuples

• We can use tuples on the right side of the string formatting operator.

• Allows us to insert multiple values into the string.

`````
"%i %i %i" % (1, 2, 3)
``````
s=???
x=10
y="Hello"
z=3.14
print s % x,y,z

a) "%i %f %s"
b) "%f %s %i"
c) "%i %s %f"
d) None of the above.