CS101: Intro to Computing
Spring 2016

Lecture 9
Administrivia

• Homework 4 is due Friday
• Midterm 1 is February 29\textsuperscript{th} at 7pm
REVIEW
s="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
t=""
i=0
while i<8:
    t=t+s[i+1]
    i+=2

What is the final value of t?

a) "ACEG"
b) "BDFH"
c) "ABCDEF"
d) "ABEF"
```python
s="0123456789"
t=""
i=0
while i<5:
    if (i%2)==1:
        t=t+s[i-1]
    else:
        t=t+s[i+1]
i=i+1
```

What is the final value of t?

a) “92143”
b) “103254”
c) “10325”
d) “921436”
LOOPING
While loop

• Allows for *repeated execution* of code
• Execute a block over and over as long as a Boolean condition is True
• *Stop executing* if Boolean condition is False
While loop

• We create an *while loop* by typing:
  1. the keyword *while*
  2. a Boolean expression
  3. a *block* of code
Accumulator pattern

• Common and useful pattern to design programs

• **Accumulator** variable keeps track of result
  – Updated in each loop iteration
Exercise

• Write a function to sum all of the digits in a number

\[ \text{sum_digits}(1214) \rightarrow 1+2+1+4 \rightarrow 13 \]
def sum_digits(n):
    s=str(n)
    i=0
    result=0
    while i<len(s):
        result=result+int(s[i])
        i=i+1
    return result
FOR LOOPS
Example

```python
i=0
while i<len(s):
    print(s[i])
    i=i+1
```
For loop

• Loop construct to make our lives easier
• Used to iterate over *iterable* types
  – Example: strings (more to come)
• Step through a sequence “one at a time”
For loop

• We create an *for loop* by typing:
  1. the keyword *for*
  2. a loop variable (just a variable)
  3. they keyword *in*
  4. an iterable
  5. a *block* of code
Example

my_string="abcdefg"
for letter in my_string:
    print(letter)
def sum_digits(n):
    result=0
    for letter in str(n):
        result=result+int(letter)
    return result
s = "abcdefg"
t = 
for c in s:
    t = c + t

What is the value of t?
a) "abcdefg"
b) "gfedcba"
c) "a"
d) "g"
s="Run The Jewels"
t=""
for c in s:
    if c.isupper():
        t=t+c.lower()

What is the value of t?
a) “RTJ”
b) “un he ewels”
c) “”
d) “rtj”
FILE INPUT
Files

- Iterable type
- Created with built-in function `open()`
- 1 argument: file name as a string (for now)
- Each item in the iterable is a string representing one line in the file

```python
for line in open("words.txt")
    print(line)
```
Example

total=0
for line in open("numbers.txt"):  
    total+=int(line)
print(total)
for w in open("words.txt"):
    vowels=0
    for c in w.lower():
        if c in 'aeiou':
            vowels+=1
    print(w.strip()+" %i" % vowels)
LISTS
Lists

• Represents an ordered collection of *items* or *elements*
  – Another *iterable* type

• Our first *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

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

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

```python
s = "Puraty Ring"
s[3] = "i"  # NOT ALLOWED
s = s[:3] + "i" + s[4:]
```
Immutable Assignment

\[ x = 3.14 \]
\[ y = x \]

Memory

\[ \text{Box } x \rightarrow 3.14 \]
\[ \text{Box } y \rightarrow 3.14 \]
Immutable Assignment

s = "Purity Ring"

Memory

\[
\begin{align*}
\text{s} & \rightarrow \text{"Purity Ring"} \\
\text{t} & \rightarrow \text{"Purity Ring"}
\end{align*}
\]
Dissimilarity to Strings

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

\[x = [4, 1, 2, 3]\]
\[x[3] = -2 \quad \leftarrow \text{item assignment}\]
\[x.\text{append}(5)\]
\[\text{del } x[1]\]
\[x.\text{sort}()\]
Mutable Assignment

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

Memory

![Diagram showing two variables, `a` and `b`, both referencing the same list `\([1, 2, 3, 4]\)` in memory.](image)
Aliasing

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

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

Memory

\[ [1, 2, 3, 2] \]
```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]
DANGER!! DANGER!!

- The `sort` and `append` methods modify the list *itself*
- This means they **RETURN NULL**

```python
x = [1, 2, 3]
new_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
Range

- The **range function** returns an immutable iterator containing integers
- Can be cast to a list
- Two arguments:
  - the starting value our range
  - the ending (not included!) value in our range

```python
x=list(range(2,5))
```
Example

total=0
for x in range(0,1000):
    total=total+x
print(total)
total=0
for x in range(0,1000):
    prime=True
    for y in range(2,x):
        if (x%y)==0:
            prime=False