Week 3
Administrivia

- Homework 4 is due Wednesday
- Homework 5 is due Friday
- Exam 1 is on July 6th
- Exam 2 is on July 22nd
What is the final value of $t$?

a) “ACEG”
b) “BDFH”
c) “ABCDEF”
d) “ABEF”
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"
FOR LOOPS
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

• 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. the 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="abcdefghijklmnopqrstuvwxyz"
t=""
for c in s:
    t=c+t

What is the value of t?

a) “abcdefghijklmnopqrstuvwxyz”
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”
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 \textit{items} or \textit{elements}
  – Another \textit{iterable} type

• Our first \texttt{container} type
  – Contains other values of \textit{any type}
  – \textbf{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)

Given:
- `s = "Puraty Ring"
- `s[3] = "i"` is NOT ALLOWED
- `s = s[:3] + "i" + s[4:]`

Immutable Assignment

\[ x = 3.14 \]

\[ y = x \]

Memory
 Immutable Assignment

s = "Purity Ring"
t = s

Memory

```
s
("Purity Ring")
t
("Purity Ring")
```
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

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

\[
b = a
\]

Memory

```
  a
  \[1, 2, 3, 4]\n  b
```
Aliasing

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

\[ a = [1, 2, 3, 4] \]
\[ b = a \]
\[ b[-1] = 2 \]
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]
print(len(x))
```

```python
x=x.append(5)
print(len(x))
```
```python
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)
Example

total=0
for x in range(0,1000):
    prime=True
    for y in range(2,x):
        if (x%y)==0:
            prime=False
    if prime:
        total+=1

print(total)
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
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
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 \textbf{string} method that operates on a \textit{list}.
- Returns a \textbf{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)
  \[ t=(1,3.14,"Hi") \]
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 assignment

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

```
one, pi, hello = (1, 3.14, "Hi")
```
- Convenient for swapping values:

```
x, y = y, x
```
s = "1,3,6,10"
a = s.split("","")
i = 1
x = 0
while i < len(a):
    r, s = a[i-1:i+1]
    x += int(s) - int(r)
i += 1

What is the final value of x?
a) 0
b) 8
c) 9
d) 11
s="1,3,6,10"
a=s.split("",")
i=1
x=0
while i<len(a):
    r,s=a[i-1:i+1]
    x+=int(s)-int(r)
i+=1

What is the final value of x?

  a) 0
  b) 8
  c) 9
  d) 11
s="1,3,6,10"

a=s.split("","")
i=1
x=0

while i<len(a):
    r,s=a[i-1:i+1]
    x+=int(s)-int(r)
    i+=1

What is the final value of x?

a) 0
b) 8
c) 9
d) 11
s="1,3,6,10"
a=s.split("","")
i=1
x=0
while i<len(a):
    r,s=a[i-1:i+1]
    x+=int(s)-int(r)
i+=1

What is the final value of x?

a) 0  
b) 8  
c) 9  
d) 11
s = "1, 3, 6, 10"
a = s.split("", ")
i = 1
x = 0
while i < len(a):
    r, s = a[i-1:i+1]
    x += int(s) - int(r)
i += 1

What is the final value of x?
a) 0
b) 8
c) 9
d) 11
s="1,3,6,10"
a=s.split("","")
i=1
x=0
while i<len(a):
    r,s=a[i-1:i+1]
    x+=int(s)-int(r)
i+=1

What is the final value of x?

a) 0
b) 8
c) 9
d) 11
s = "1,3,6,10"
a = s.split(",")
i = 1
x = 0

while i < len(a):
    r, s = a[i-1:i+1]
    x += int(s) - int(r)
    i += 1

What is the final value of x?

a) 0  
b) 8  
c) 9  
d) 11
s="1,3,6,10"
a=s.split("","\n"")
i=1
x=0
while i<len(a):
    r,s=a[i-1:i+1]
    x+=int(s)-int(r)
    i+=1

<table>
<thead>
<tr>
<th>i_{old}</th>
<th>x_{old}</th>
<th>r</th>
<th>s</th>
<th>x_{new}</th>
<th>i_{new}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>6</td>
<td>10</td>
<td>9</td>
<td>4</td>
</tr>
</tbody>
</table>
Tuple 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.
Tuples and iteration

• `zip` - iterate through two iterables together
• Loop variable assigned a series of tuples

```python
x=[1,2,3,4]
y="ABCD"
for a in zip(x,y):
    print a
```
Tuples and iteration

- enumerate - count as we iterate
- Loop variable contains a tuple

```python
x = "ABCD"
for a in enumerate(x):
    print a
```
Exercises

1. Find all of the palindromes in words.txt
2. Find the longest palindrome in words.txt
3. Find longest word that uses only two letters.
4. Find and sort all of the even numbers in numbers.txt
def palindrome(word):
    bword=""
    for c in word:
        bword=c+bword
    return bword==word

longest_p=""
longest_l=0
for word in open("words.txt"):
    word=word.strip().lower()
    if palindrome(word):
        if len(word)>longest_l:
            longest_p=word
            longest_l=len(word)

print longest_p
s = [1, 2, 3]
t = [4, 5, 6]

s[0], s[1] = t[1], t[2]

What is the final value of s?

a) [1, 4, 5]
b) [4, 5, 3]
c) [5, 6, 3]
d) [1, 5, 6]
def fun(a, b):
    return b, a

x = fun(1, 2)[0]

What is the final value of x?

a) (1, 2)
b) (2, 1)
c) 1
d) 2
TUPLES AND ITERATION
Zip

• zip - iterate through two iterables together
• Loop variable assigned a series of tuples
  
x=[1,2,3,4]
y="ABCD"

for a in zip(x,y):
    print(a)
```python
s="WTE"
t="ANY"
u=[]
for a,b in zip(s,t):
    u.append(a+b)
x=''.join(u)
```

What is the final value of x?

a) ""
b) "AWNTYE""
c) "WTEANY""
d) "WATNEY""
Enumerate

- enumerate - count as we iterate
- Loop variable contains a tuple

```python
x="ABCD"
for a in enumerate(x):
    print(a)
```
s="234"

x=0

for a,b in enumerate(s):
    x+=int(b)+a

What is the final value of x?

a) 12
b) 237
c) 9
d) 15
DICTIONARIES
Types we’ve learned

• Lists and tuples are *ordered*
• Lists and tuples are *indexed using a integers*
• It’s natural to associate data with an identifier:
  – Person → birthday, gender, parents
  – Country → flag, median income
Mustang → Ford
Corvette → Chevrolet
Civic
Accord
Accent
Golf → Volkswagen

Model
Make
Dictionaries

• Represents an *unordered* collection of items or elements

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

• Can be indexed with *any type*

• Map *keys* to *values*
Dictionary literals

• We create an *dictionary* by typing:
  1. an open curly bracket `{`
  2. a key, a colon, and its associated value
  3. key:value pairs separated by commas
  4. a closing curly bracket `}`
Dictionary

model=
{"Civic":"Honda",
"Mustang":"Ford",
"Corvette":"Chevy",
"Accord":"Honda",
"Accent":"Hyundai"}
Dictionary Operations

d={"one":1,"two":2,"three":3}
print(d["one"])
d["four"]=4
del d["four"]
"five" in d
for key in d:
    print(key,d[key])
d={"a":2,"c":3,"b":1}
x=d["a"]+d["c"]

What is the final value of x?

a) 3
b) "ac"
c) 4
d) 5
d={}
for i,j in enumerate("ABC"):
    d[j]=i

What is the final value of d?
a) {"A":0,"B":1,"C":2}
b) {0:"A",1:"B",2:"C"}
c) {"A":1,"B":2,"C":3}
d) {1:"A",2:"B",3:"C"}
d = {}
for i, j in zip("WAT", "NEY"):  
    d[(i, j)] = j

Which of these expressions evaluates to "E"?

a) d["A"]
b) d["E"]
c) d["A", "E"]
d) d["EA"]
USES FOR DICTIONARIES
Dictionaries to Encode

- We can use dictionaries to encode/decode data
- We can use dictionaries to translate from one representation to another
Dictionaries to Encode

```python
x="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
y="BCDEFGHIJKLMNOPQRSTUVWXYZA"
e={}
for i in range(len(x)):
    e[x[i]]=y[i]
encoded=""
for c in "HELLO":
    encoded+=e[c]
```
Dictionaries to Encode

```python
x = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
y = "BCDEFGHIJKLMNOPQRSTUVWXYZA"
d = {}
for i in range(len(x)):
    d[y[i]] = x[i]
decoded = ""
for c in encoded:
    decoded += d[c]
```
Exercise

• Encode all of the words in a file with the Caesar cypher
• Decode all of the words in the file
Dictionaries as Accumulators

- We can use dictionaries as a collection of counters for many things at once

```python
x = "ABBACAB"
d = {}
for c in x:
    if c not in d:
        d[c] = 0
    d[c] += 1
```
Exercise

• Count category frequencies in Jeopardy questions
• Count bigram frequencies in Jeopardy clues
Dictionaries to Join/Merge Data

• We can link data based on a common field
zip={'Bill':60644,  
    'Jim':41073,'Beth':63103}
city={60644:"Chicago",  
     41073:"Cincinnati",  
     63103:"St. Louis"}
for name in zip:  
    print(name,city[zip[name]])
Modules

• A collection of Python specialized functions, variables, and even types
• We need to **import** the module
  ```python
  import math
  ```
• Can then access things within the module using **attribute operator**
  ```python
  math.sqrt(math.pi)
  ```
From

- Can choose what to import with `from`:

  ```python
  from cmath import phase
  phase(1+1j)
  ```
Writing readable code

• We should always strive to write code that is easy to read.
  – Our variables should have descriptive names.
  – We should also annotate our code.

• **REMEMBER**: A program is set of instructions a computer executes to achieve a goal.
Commenting

• **Comments** are text that the interpreter ignores
• Comments help *a person* read a program
• The # symbol indicates a comment
  – Anything after that symbol is ignored
# Hello, I am a comment
Docstring

• A string literal that behaves like a comment
• Use triple quotes
• Especially useful after function definition

"""Hello, I am a docstring."""
s = "ABCD"
s += "1"
#s += "2"
""""s += 3""""

What is the final value of x?

a) ABCD
b) ABCD1
c) ABCD12
d) ABCD123
Why comment/document?

• Allows us to *explain* our code to others.
• But mostly… to ourselves.
• Yes, *ourselves.*
Every script (.py) file you write is a module.

Your modules should have a docstring at the beginning describing them and you.

```
CS101 class demonstration
Author: Ryan Cunningham
```

```
Documenting Functions

- Use doc string and describe what function does.
- Describe all parameters by name.
- Describe all return values.
Main function

• Allows our module to be imported OR run from the command line as a script
• Put the “starting point” code in a function called “main”
• This test checks if running on command line:

```python
if __name__ == '__main__':
    main()
```
s="WTE"
t="ANY"
u={}
for a,b in zip(s,t):
    u[a]=b
x=u["T"]

What is the final value of x?
a) “A”
b) “Y”
c) “T”
d) “N”
s=[4,8,15,16,23,42]
d={0:0,1:0}
for a,b in enumerate(s):
    d[a%2]+=1
x=d[1]

What is the final value of x?
a) 0
b) 1
c) 2
d) 3
USES FOR DICTIONARIES
Dictionaries to Encode

- We can use dictionaries to encode/decode data
- We can use dictionaries to translate from one representation to another
Exercise

• Encipher all of the words in a file with the Caesar cypher
• Substitute each letter with the next in the alphabet

HELLO ➔ IFMMP
def encypher(word):
    word=word.upper()
    x="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    y=x[1:]+x[0]
    e={}
    for a,b in zip(x,y):
        e[a]=b
    encoded=""
    for c in word:
        if c in encoded:
            encoded+=e[c]
        else:
            encoded+=c
    return encoded

for line in open("words.txt"):
    line=line.strip()
    print(encypher(line))
• We can use dictionaries as a collection of counters for many things at once

```python
x = "ABBACAB"
d = {}
for c in x:
    if c not in d:
        d[c] = 0
    d[c] += 1
```
Counting bigrams

counter={}  
for word in open("words.txt"):  
    word=word.strip().upper()  
    for i in range(len(word)-1):  
        bi=word[i:i+2]  
        if bi not in counter:  
            counter[bi]=1  
        else:  
            counter[bi]+=1
Exercise

• Count category frequencies in Jeopardy questions
Dictionaries to Join/Merge Data

• We can link data based on a common field

    `zip={"Bill":60644,
              "Jim":41073,"Beth":63103}

    `city={60644:”Chicago”,
              41073:”Cincinnati”,
              63103:”St. Louis”}

    `for name in zip:
        print(name,city[zip[name]])`
Exercise

• Print the album, artist, and track names for each song in tracks.csv
import csv

# read in artist data from csv file
artist_file = open("artist.csv")
artist_d = {}
artist_reader = csv.DictReader(artist_file)
for artist in artist_reader:
    artist_id = artist["ArtistId"]
    name = artist["Name"]
    artist_d[artist_id] = name

artist_file.close()
# read in album data from csv file
album_file=open("album.csv")
album_reader=csv.DictReader(album_file)
album_d={}
for album in album_reader:
    album_id=album["AlbumId"]
    album_t=album["Title"]
    artist=album["ArtistId"]
    # store artist/album as a tuple
    album_d[album_id]=(album_t,artist)

album_file.close()
#read track data and merge with album
# and artist data to print it out
track_file=open("track.csv")
track_reader=csv.DictReader(track_file)
for track in track_reader:
    name=track["Name"]
    album_id=track["AlbumId"]
    title,artist_id=album_d[album_id]
    artist=artist_d[artist_id]
    print name","",title","",artist

track_file.close()
s="ABACAB"
d={}
for a,b in enumerate(s):
    if b not in d:
        d[b]=0
    d[b]+=a

da) d={1:"A",3:"C","2:"B"}
b) d={"A":3,"B":2,"C":1}
c) d={"A":6,"B":6,"C":3}
d) d={"A":0,"B":0,"C":0}
s="ABACAB"
d={}
for a,b in enumerate(s):
    if b not in d:
        d[b]=a
    else:
        d[b] +=a
MODULES
Modules

- A collection of Python specialized functions, variables, and types
- We need to \textit{import} the module
  \begin{verbatim}
  import math
  \end{verbatim}
- Can then access things within the module using \textit{attribute operator}
  \begin{verbatim}
  math.sqrt(math.pi)
  \end{verbatim}
From

• Can choose what to import with `from`
  
  ```python
  from cmath import phase
  phase(1+1j)
  ```

• We don’t have to type the module name all the time

• Import multiple items with a comma
  
  ```python
  from cmath import phase, rect
  ```
math.degrees(2*pi)

What should replace the ???

a) from math import pi
   import math

b) from math import pi, degrees

c) import pi
   import math

d) import math
exp(pi)−pi

What should replace the ???

a) from math import pi
    import math
b) from math import pi, exp
c) import pi, exp
    import math
d) import math
Useful Python Modules

- math, cmath
- random
- csv
- sys, os
- time, datetime
- itertools
- logging
- NumPy
- SciPy
- matplotlib
WRITING CODE
Writing readable code

- We should always strive to write code that is easy to read.
  - Our variables should have *descriptive* names.
  - We should also *annotate* our code.

- **REMEMBER**: A program is set of instructions a computer executes *to achieve a goal.*
Commenting

- **Comments** are text that the interpreter ignores
- Comments help *a person* read a program
- The # symbol indicates a comment
  - Anything after that symbol is ignored

# Hello, I am a comment
Docstring

• A string literal that behaves like a comment
• Use triple quotes
• Especially useful after function definition

"""Hello, I am a docstring."""
s="ABCD"
s+="1"
#s+="2"
""""s+=3"""

What is the final value of x?
a) ABCD  
b) ABCD1  
c) ABCD12  
d) ABCD123
Why comment/document?

• Allows us to *explain* our code to others.
• But mostly... to ourselves.
• Yes, *ourselves*. 
Documenting Modules

• Every script (.py) file you write is a module.
• Your modules should have a docstring at the beginning describing them and you.

"""
CS101 class demonstration
Author: Ryan Cunningham
"""
Documenting Functions

• Use doc string and describe what function does.
• Describe all parameters by name.
• Describe all return values.

def sqrt(n):
    """
    Computes square root of a number.
    n: an integer or float
    returns: the square root of n
    """
    return n**.5
category_counts={}  # accumulator for counting categories
for line in open("jeopardy.txt"):
    if line[0]!="#":  # ignore comment lines
        split=line.split("\t")  # split on tab
        category=split[2]  # category is 3rd column
        # increment count of this category
        if category not in category_counts:
            category_counts[category]=0
        category_counts[category]+=1

# sort category counts for display
category_tuples=[]
for c in category_counts:
    n=category_counts[c]
    category_tuples.append((n,c))
category_tuples.sort()  # tuples are sorted by first item
print category_tuples