import aid

# sizes
i1 = [1, 'x']
i2 = [2, 'y']
result = aid.common(i1, i2)
expected = []
if result != expected:
    print('given', i1, 'and', i2, 'gave', result, 'not', expected)

i1 = [1, 'x', 'w']
i2 = [2, 'y']
result = aid.common(i1, i2)
expected = []
if result != expected:
    print('given', i1, 'and', i2, 'gave', result, 'not', expected)

i1 = [1, 'x']
i2 = [2, 'y', 'w']
result = aid.common(i1, i2)
expected = []
if result != expected:
    print('given', i1, 'and', i2, 'gave', result, 'not', expected)

i1 = [-1.0, 'y']
i2 = [2.3, 'fun']
result = aid.common(i1, i2)
expected = []
if result != expected:
    print('given', i1, 'and', i2, 'gave', result, 'not', expected)

i1 = [[1,2], 'y']
i2 = ['fun', 4, '3']
result = aid.common(i1, i2)
expected = []
if result != expected:
    print('given', i1, 'and', i2, 'gave', result, 'not', expected)

i1 = [1, 2, 3]
i2 = [1, 2, 3]
result = aid.common(i1, i2)
expected = [1, 2, 3]
if result != expected:
    print('given', i1, 'and', i2, 'gave', result, 'not', expected)

i1 = [2, 1, 3, 5]
i2 = [1, 2, 3, -1, 3]
result = aid.common(i1, i2)
expected = [2, 1, 3]
if result != expected:
    print('given', i1, 'and', i2, 'gave', result, 'not', expected)

# empty
result = aid.common([], [1, 'x'])
result = aid.common([1, 'x'], [])
result = aid.common([], [])
'Purpose: list manipulation

```python
def greater( x, k ) :
    ''' Return all of the elements in x that are greater than k
    '''
    g = []
    for item in x:
        if item > k:
            g.append(item)
    return g

def inorder( x ) :
    ''' Return whether list x’s elements are in non-decreasing sorted order
    '''
    # y = x[:]
    # x.sort() # BAD - modifies the list
    # return x == y

    # return x == sorted(x) - Good; does same thing as longer code below

    previous = x[0]
    all_sorted = True
    for i in range(len(x)):
        if previous <= x[i]:
            previous = x[i]
        else:
            all_sorted = False
            break
    if all_sorted:
        return True
    else:
        return False

def duplicate( x ) :
    ''' Return a new list whose elements are the same as x’s elements
    '''
    # return x[:] # works
    # return x # is not a new list - does not work

    y = []
    for element in x:
        y.append(element)
    return y
```
'Purpose: string cleaning

# helpful named strings
PUNCTUATION = '!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~'
WHITE_SPACE = ' 	

EXTRANEOUS = PUNCTUATION + WHITE_SPACE

def lower( list ):
    ''' Returns a new list whose elements are lower case versions of those in list '''

    ans = []
    for word in list:
        small = word.lower()
        ans.append( small )
    return ans

def remove_extras( word ):
    ''' Returns a new version of string where the leading and trailing extraneous characters (punctuation or whitespace) have been removed '''

    return word.strip(EXTRANEOUS)

def strip( list ):
    ''' Returns a new version of list where the corresponding elements have leading and trailing extraneous characters (punctuation or whitespace) removed '''

    ans = []
    for word in list:
        better = remove_extras( word )
        ans.append( better )
    return ans
'Purpose: develop functional problem solving skills given problem descriptions

```python
def compare( a, b ):
    '''Function compare() requires parameters a and b.
    It returns
    1, if a is greater than b;
    0, if a is equal to b; and
    -1, if a is less than b'''
    # three cases - implies if-elif-else
    if ( a > b ) :
        # returns 1, if a is greater than b
        return 1
    elif ( a == b ) :
        # returns 0, if a is equal to b
        return 0
    else : # a < b
        # returns -1, if a is less than b
        return -1

def sum_of_powers_of_two( n ) :
    '''Function sum_of_powers_of_two() uses its non-negative integer parameter n
to determine the number of terms in the summation it computes.
The function returns the sum of 2**0 + 2**1 + â\200¦ + 2**n'''
    # return 2**( n+1 )-1
    sum = 0
    for number in range( 0, n+1 ) :
        current = 2**number
        sum = sum + current
        # power = 0
        # while ( power <= n ) : #
        #     sum = sum + ( 2 ** power )
        #     power = power + 1
    return sum

import math

def factorial( n ) :
    '''Function factorial() computes and returns the value of n! (read n-factorial)
    for its integer non-negative parameter n.
    That is, if n is 0, the function returns 1; otherwise, the function returns 1â•2â• â\200¦ Â• n'''
    # return math.factorial( n )
    # Maybe we should use Sterling’s formula? Nah, that sounds like work...
    if ( n == 0 ) :
        return 1
    product = 1
    for i in range( 1, n+1 ) :
        product = product * i
    return product

def lenmax( s ) :
    '''Function lenmax() requires a non-empty string list parameter s, and displays the longest
```python
t string in s'''
    longest = ''  # or s[0], which might be safer in general (both ok for this task)...
    for word in s :
        if ( len( word ) > len( longest ) ) :
            longest = word
    print( longest )

def cross( a, b ) :
    '''Function cross( ) requires two non-empty numeric list parameters a and b. The function returns None, if and a and b have different lengths; otherwise, it returns ( a[ 0 ] * b[ 0 ] ) + ( a[ 1 ] * b[ 1 ] ) + \( \cdots + ( a[ n-1 ] * b[ n-1 ] ) \).'''
    if ( len( a ) != len( b ) ) :
        return None
    sum = 0
    # some kind of loop goes here... only one loop
    n = len( a )
    for i in range( 0, n ) :
        v = a[i] * b[i]
        sum = sum + v
        # sum = sum + a[i] * b[i]
    return sum

def lenmax( list_of_strings ) :
    pass

def multiply( a ) :
    pass

def same( a, b ) :
    pass

def roots( a, b, c ) :
    pass

if __name__ == "__main__" : # module testing
    # this magical statement means "only run this code if this file is the one being run"
    print( cross( [1, 2, 3], [1, 2, 3, 4] ), 'should be None' )
    print( cross( [1, 2], [3, 4] ), 'should be', ( 1*3 )+( 2*4 ) )
    print( cross( [1, 2, 3.4, -5.1], [3, 4, 5, 5.6] ), 'should be', ( 1*3 )+( 2*4 )+( 3.4*5 )+( -5.1*5.6 ) )

    # print( compare( 3.4, 4.8 ), 'should be -1' )
    # print( compare( 3, 4 ), 'should be -1' )
    # print( compare( 4, 3 ), 'should be +1' )
    # print( compare( 3, 3 ), 'should be 0' )
    # print( compare( -3, -4 ), 'should be +1' )
    # print( compare( 'alpha', 'beta' ), 'should be -1' )
    # print( compare( 'alpha', 3 ), 'should be an error' )
    #
    # print( sum_of_powers_of_two( 0 ), 'should be 1' )
    # print( sum_of_powers_of_two( 1 ), 'should be', 1 + 2 )
    # print( sum_of_powers_of_two( 2 ), 'should be', 1 + 2 + 4 )
    # print( sum_of_powers_of_two( 3 ), 'should be', 1 + 2 + 4 + 8 )
    # print( sum_of_powers_of_two( 4 ), 'should be', 1 + 2 + 4 + 8 + 16 )
    #
    # print( factorial( 0 ), 'should be 1' )
```
# print( factorial( 10 ), 'should be', 10*9*8*7*6*5*4*3*2*1 )
#
# lenmax( ['hi', 'there'] )
# print( 'the output line above this line should be "there"' )
# purpose: demonstrate use of a while statement by generating the first
# n primes, for a user-specified n

import math  # need to be able to calculate square roots

# get the n of interest
reply = input('Enter number of primes to be generated: ')

n = int(reply)

# initialize i to be the first candidate prime
i = 2

# initialize found to be the number of primes generated so for
found = 0

# loop while we have not found n primes
while (found < n):
    s = math.sqrt(i)
    s = int(s)

    # keep track whether we have a factor for candidate i; that is,
    # check out possible factors 2 .. s
    is_prime = True
    for number in range(2, s + 1):
        remainder = i % number
        is_a_factor = remainder == 0
        if (is_a_factor is True):
            is_prime = False

    if (is_prime is True):
        found = found + 1
        print(i, 'is prime

    # move candidate i along
    i = i + 1

# all done
# purpose: demonstrate use of a while statement by generating the first
#          n primes, for a user-specified n

import magic

# get the n of interest
reply = input('Enter number of primes to be generated: ')

n = int(reply)

# initialize i to be the first candidate prime
i = 2

# initialize found to be the number of primes generated so far
found = 0

# loop while we have not found n primes
while (found < n):
    is_prime = magic.is_prime(i)
    if (is_prime is True):
        found = found + 1
        print(i, 'is prime')

    # move candidate i along
    i = i + 1

# all done
# purpose: show use of a dictionary for problem solving

# get access to url support
import tools

# CS 1112 translation dictionary -- entries are of form: word translation
BABEL_FISH_URL = 'http://www.cs.virginia.edu/~cs1112/words/dictionaries/babel'

# get contents of babel fish url
text = tools.read_web( BABEL_FISH_URL )
text = text.strip()
entries = text.split( '\n' )

# put entries into a dictionary - a mapping of words to translations
# the dictionary starts off empty
dictionary = {}

# add each word-translation pair to the dictionary
for entry in entries :
    # split entry into components and add mapping to the dictionary
    if ( entry == '' ) :
        continue
    f, e = entry.split()
dictionary[ f ] = e

# now have our words and translations - next up the user text to translate
reply = input( 'Enter phrase to be translated: ' )

print()

# convert reply to list of words to be translated
to_be_translated = reply.split()

# translate the words one by one
for word in to_be_translated :
    # to translate the word must be known
    if ( word in dictionary ) :
        # its there, so we can get its translation
e = dictionary[ word ]
    else :
        # not there, so translation is word surrounded by delimiters > and <
e = '>\' + word + '<\'

    # print current word followed by a space'
    print( e, end= ' ' )

# all done
print()
# task: transform a line of text into a line of correctly spelled words

```python
import tools

# link to correct spellings
CORRECT_SPELLINGS = 'http://www.cs.virginia.edu/~cs1112/datasets/words/spellings'

# read the page
text = tools.read_web( CORRECT_SPELLINGS )

# list of correctly spelled words
#   -- get rid of leading and trailing whitespace
#   -- split words on internal whitespace
spellings = text.strip().lower().split()

# get the user text as a list of words
reply = input( 'Enter text: ' )
user_words = reply.strip().split()

# find and correct typos

# initialize accumulator of correct spelled words
corrected = ''

# examine the words one by one
for word in user_words :
    # get standard form of word
    standard = word.lower()
    # check if its there
    if (standard in spellings) is True:
        have_correct = True
    else:
        have_correct = False

    while (have_correct is False):
        word = input(word + ' is misspelled; what should it be? ')
        standard = word.lower()
        have_correct = (standard in spellings) # could be lines 30-33 here instead; does same
        thing

    corrected = corrected + word + '

# strip the correction of trailing whitespace
corrected = corrected.strip()

# print the result
print( 'Corrected version:', corrected )
```
import math  # for sqrt

def is_prime(i):
    # check whether candidate i is prime. it is prime if it has no
    # factors other than itself and 1
    #
    # observation: if candidate i is not prime, it has a factor f
    # between 2 .. s (inclusive), where s is the sqrt( i )
    s = math.sqrt(i)
    s = int(s)

    # keep track whether we have a factor for candidate i; that is,
    # check out possible factors 2 .. s
    is_prime = True
    found_a_factor = False
    for number in range(2, s + 1):
        remainder = i % number
        is_a_factor = remainder == 0
        if (is_a_factor):
            found_a_factor = True
            is_prime = False

    # we now know whether candidate i is prime
    return is_prime
import math  # for sqrt

i = input( 'Type a number: ' )
i = int(i)

# check whether candidate i is prime. it is prime if it has no
# factors other than itself and 1
#
# observation: if candidate i is not prime, it has a factor f
# between 2 .. s (inclusive), where s is the sqrt( i )
s = math.sqrt(i)
s = int(s)

# keep track whether we have a factor for candidate i; that is,
# check out possible factors 2 .. s
is_prime = True
found_a_factor = False
for number in range( 2, s + 1 ):
    remainder = i % number
    is_a_factor = remainder == 0
    if ( is_a_factor ) :
        found_a_factor = True
        is_prime = False

# we now know whether candidate i is prime
print(is_prime)
# purpose: demonstrate while loop and dict mappings

# get url reading abilities

import tools

# where is the dog to notoriety mappings

PUPPIES_CSV = 'http://www.cs.virginia.edu/~cs1112/datasets/puppies.csv'

# get contents of url

text = tools.read_web( PUPPIES_CSV )

# convert text into dog to claim-to-fame mappings

lines = text.strip().split( '
' )

dogs = {}
for line in lines:
    name, fame = line.split( ',' )
    # key     value
    # print('before:', dogs)
    # print('dogs[', name, ']=', fame)
    dogs[name] = fame
    # print('after:', dogs)
    # input('Type enter to continue')
    # you can only have one value per key...

print(dogs)
print(dogs.keys())
print(dogs.values())

# get user query
reply = input('What dog do you care about? ')

# analyze the query
if reply in dogs.keys():
    fame_of_reply = dogs[reply]
    print(reply, 'is best known from', fame_of_reply)
else:
    print('I know nothing about', reply)

# print result of determination

# all done
# purpose: show use of a dictionary for problem solving

# get access to url support
import tools

# CS 1112 translation dictionary -- entries are of form: word translation
BABEL_FISH_URL = 'http://www.cs.virginia.edu/~cs1112/words/dictionaries/babel'

# get contents of babel fish url
text  = tools.read_web( BABEL_FISH_URL )
text  = text.strip()
entries = text.split( '\n' )

# put entries into a dictionary - a mapping of words to translations
# the dictionary starts off empty

dictionary = {}

# add each word-translation pair to the dictionary
for entry in entries :
    # split entry into components and add mapping to the dictionary
    if ( entry == '' ) :
        continue
    f, e = entry.split()
    dictionary[ f ] = e

print( dictionary )

# now have our words and translations - next up the user text to translate
reply = input( 'Enter phrase to be translated: ' )

print()

# convert reply to list of words to be translated
to_be_translated = reply.split()

# translate the words one by one
for word in to_be_translated :
    # to translate the word must be known
    if ( word in dictionary ) :
        # its there, so we can get its translation
        e = dictionary[ word ]
    else :
        # not there, so translation is word surrounded by delimiters > and <
        e = '>' + word + '<'

    # print current word to output. observe we can specify what happens after
    # the output appears
    print( e, end=' ')

# all done
print()
A module of helper tools for CS 1112.

Contains:

- `read_web` - returns the contents of a webpage as a string
- `distance` - computes distance in miles between points on earth

Usage:

```python
import tools
data = tools.read_web('http://some-link.goes/here')
miles = tools.distance(38.03, -78.2, 0, 0)
```

```python
import urllib.request
import math
def read_web(link):
    '''read_link(url)
    Will go online and retrieve the contents of the given url,
    converting the answer to a string and returning it.
    '''
    stream = urllib.request.urlopen(link)
    page = stream.read()
    text = page.decode('UTF-8')
    return text

def distance(lat1, long1, lat2, long2):
    ''' Computes distance in miles from (lat1, long1) to (lat2, long2)
    '''
    EARTH_RADIUS = 3958.7621
    # convert latitude and longitude into radians
    degrees_to_radians = math.pi / 180.0
    phi1 = (90.0 - lat1) * degrees_to_radians
    theta1 = long1 * degrees_to_radians
    phi2 = (90.0 - lat2) * degrees_to_radians
    theta2 = long2 * degrees_to_radians
    # calculate spherical distance for unit sphere
    cos = (math.sin(phi1) * math.sin(phi2) * math.cos(theta1 - theta2) +
           math.cos(phi1) * math.cos(phi2))
    unit_sphere_distance = math.acos(cos)
    # scale distance by indicated radius
    distance = unit_sphere_distance * EARTH_RADIUS
    # got the distance, so hand it back
    return distance
```
# purpose: show synced lists for problem solving

# get access to url support
import urllib.request

# CS 1112 translation dictionary -- entries are of form: word translation
BABEL_FISH_URL = 'http://www.cs.virginia.edu/~cs1112/words/dictionaries/babel

# get contents of babel fish url
stream = urllib.request.urlopen( BABEL_FISH_URL )
page = stream.read()
decoding = page.decode( 'UTF-8' )
text = decoding.strip()
entries = text.split( '
' )

# convert the entries into two lists: words and translations
# the lists start off empty
english_words = []
foreign_words = []

# add each word-translation pair to the lists
for entry in entries :
    # split entry into components and add to appropriate lists
    f, e = entry.split()
    foreign_words.append( f )
    english_words.append( e )

# now have our words and translations - next up the user text to translate
reply = input( 'Enter phrase to be translated: ' )

print()

# convert reply to list of words to be translated
to_be_translated = reply.split()

# translate the words one by one
for word in to_be_translated :
    # to translate the word must be known
    if ( word in foreign_words ) :
        # finds its spot in the list, because its translation is in the
        # corresponding spot in the other list
        i = foreign_words.index( word )
        e = english_words[ i ]
    else :
        # not there, so translation is word surrounded by > and <
        e = '>'+word+'<'

    # print current word to output. observe we can specify what happens after
    # the output appears
    print( e, end=' ' )

# all done
print()
# purpose: play guess a number
# human picks the number; computer guesses it
# use while, not for
import random

# specify max number
n = 1000

# start the game
print( 'Think of an integer from 0 to', n )

# ???
at_least = 0
no_more_than = n
looking = True
while looking:
    print( '    debugging output:', at_least, no_more_than)
    number = (at_least + no_more_than + 1) // 2
    answer = input( "Is your number smaller than " + str(number)+"? " )
    if 'yes' == answer:
        no_more_than = number - 1
    else:
        at_least = number
    if at_least == no_more_than:
        looking = False

print('Found your number! It was', number)
# purpose: demonstrate loops with embedded decisions

# task: find the word of the longest length amongst the input

# get the words

reply = input( 'Enter a series of words: ' )

list = reply.split()

# need a guess for the word of longest length.  guess should never
# conflict with the words of longest length

longest_word_so_far = ''

# consider the words in the list, one by one

for word in list :

    # check the word out
    nw = len( word )
    nl = len( longest_word_so_far )

    if ( nw > nl ) :
        # current word is longer than the longest word so far, so need to update
        longest_word_so_far = word

# longest_word_so_far must be longest as we considered every word as a possibility

print( longest_word_so_far )
# purpose: demonstrate loops with embedded decisions using URL-based data set
# task: spell check a line of input
# import the url module
import urllib.request

# define URL for spelling dictionary
CS1112_DATA_SET_FOLDER = 'http://www.cs.virginia.edu/~cs1112/datasets/'
CS1112_WORDS_DATA_SET_FOLDER = CS1112_DATA_SET_FOLDER + 'words/
CS1112_SPELLING_DICTIONARY   = CS1112_WORDS_DATA_SET_FOLDER + 'spellings'

# get the spelling dictionary
# establish connection from program to web resource
stream = urllib.request.urlopen( CS1112_SPELLING_DICTIONARY )
# get the contents of the page
contents = stream.read()
# decode the contents of the page to plain text
text = contents.decode( 'UTF-8' )
# split the test into a list of word spellings
spellings = text.split()
# when debugging/developing might want to print out got the spellings

# get the user text as a list of words
reply = input( 'Enter text: ' )
user_words = reply.split()
# when debugging/developing might want to print out got the user_words

# consider the user words one by one
for word in user_words :
    # check the word out
    if ( word not in spellings ) :
        # mis-spelled so print it
        print( word )
    # else :                        # no else is necessary. if the word
    #    pass                       # is there, no action is needed
# all done
# purpose output three integer inputs in sorted order

# get the numbers
reply = input('Enter three numbers: ')

n1, n2, n3 = reply.split()

n1 = int(n1)
n2 = int(n2)
n3 = int(n3)

# determine their sorted order. there are six possibilities
if (n1 <= n2 <= n3):
    # possibility 1
    sorted1, sorted2, sorted3 = n1, n2, n3
elif (n1 <= n3 <= n2):
    # possibility 2
    sorted1, sorted2, sorted3 = n1, n3, n2
elif (n2 <= n1 <= n3):
    # possibility 3
    sorted1, sorted2, sorted3 = n2, n1, n3
elif (n2 <= n3 <= n1):
    # possibility 4
    sorted1, sorted2, sorted3 = n2, n3, n1
elif (n3 <= n1 <= n2):
    # possibility 5
    sorted1, sorted2, sorted3 = n3, n1, n2
elif (n3 <= n2 <= n1):
    # must be possibility 6
    sorted1, sorted2, sorted3 = n3, n2, n1
else:
    # if we get to the else, then our analysis was wrong and
    # needs to be rethought
    print('error in logic')

# print the sorted order
print(sorted1, sorted2, sorted3)

# all done
# A robot encounters a door. If the door is unlocked, the robot should  
# be instructed to open the door and enter the room. Note, before  
# entering the robot should determine whether the light is off. If  
# it is off, the robot should be instructed to first turn on the light.  
# If the door is instead locked, the robot should be instructed to turn  
# around

# get status of the door  
reply = input( 'Door (locked / unlocked): ' )

# put in standard format -- all lowercase  
door_sensor = reply.strip().lower()

print()

# based on door sensor take action  
if ( door_sensor == 'unlocked' ) :  
    # we can go in  
    print( 'Open the door
' )

    # need to get status of the light  
    reply = input( 'Light (on / off): ' )  
    light_sensor = reply.strip().lower()

    print()

    if ( light_sensor == 'off' ) : # we call this if statement "nested"  
        # its off so turn it on  
        print( 'Turn on the light
' )

        # can enter because the door is open and the light is on  
        print( 'Enter the room
' )

    else :  
        # cannot enter -- the door is locked  
        print ( 'Turn around
' )

else :  
    # all done
text = input('Type words')
words = text.split()

# task: find max word without using sort() or max()
last_word = words[0]  # guess what the most-X value might be
for word in words:  # check all values
    if word > last_word:  # if they are more-X, then
        last_word = word  # remember them instead of the guess

print('The alphabetically last word was', last_word)
# determine whether a list of numbers is in sorted order

# get list of numbers
reply = input('Enter numbers: ')

list = reply.split()

# determine length of list
n = len(list)

# update each element in the list to be int
for i in range(0, n):
    x = list[i]
    x = str(x)
    list[i] = x

# determine list is in sorted order. it is
# list[0] <= list[1]
# list[1] <= list[2]
# ...
#
# consider each pair starting point in the list

# assume list is sorted, check for a contradiction
is_sorted = True

for i in range(0, n-1):
    a = list[i]
    b = list[i+1]

    # determine relationship between values
    if a > b:
        # even a single out of order means the list is not sorted
        is_sorted = False

        # get out of the loop
        break
    else:
        # what action should we take

# print our determination
print(is_sorted)

# all done
numbers = [2, 1, 6, 4]

all_even_so_far = True
for number in numbers:
    this_even = (number % 2) == 0
    if not this_even:
        all_even_so_far = False
        break  # <- means "stop looping now"

if all_even_so_far == True:
    print('they were all even')
else:
    print('they were not all even')
# report message based on day of week
#       Sunday: weekend day
#     Monday: start of workweek
#       Tuesday: workday
#   Wednesday: workday
#    Thursday: workday
#     Friday: end of workweek
#   Saturday: weekend day

# get day of interest
reply = input('Enter day of week: ')

day = reply.strip().capitalize()

# analyze and report on day
if (day == 'Sunday') or (day == 'Saturday'):
    print('weekend day')
elif (day == 'Monday'):
    print('start of workweek')
elif (day == 'Friday'):
    print('end of workweek')
else:
    print('workday')
import tools

folder = 'http://www.cs1112.ninja/datasets/csv/

f = input('Input: ')
c = input('Input: ')
c = int(c)

url = folder + f

page = tools.read_web(url).strip()

rows = page.split('
')

sum = 0

for row in rows:
    entries = row.split(',,')
    entry = entries[c]
    entry = float(entry)
    sum = sum + entry

print(sum)
a = input('Input: ')
b = input('Input: ')

a, b = int(a), int(b)

print(min(a, b))
print(max(a, b))
```
import random

s = input('Input: ')
b = input('Input: ')

s, b = int(s), int(b)

random.seed(s)

for i in range(0, 5):
    nbr = random.randrange(b)
    print(nbr)
```
```python
import math

ans = 3 / 4 ; print( ans , type( ans ) )
ans = 3 % 4 ; print( ans , type( ans ) )
ans = 3 * 4 ; print( ans , type( ans ) )
ans = math.sqrt( 4 )    ; print( ans , type( ans ) )
ans = '12' + '3'       ; print( ans , type( ans ) )
ans = '12' * 3 ; print( ans , type( ans ) )
ans = 3 // 4    ; print( ans , type( ans ) )
ans = len( '12' )   ; print( ans , type( ans ) )

x = 'a b c d'
ans = x.split() ; print( ans , type( ans ) )
ans = list( 'abcd' ) ; print( ans , type( ans ) )

x = [ '2', '8', '128' ]
ans = max( x )  ; print( ans , type( ans ) )

x = 'procrastination'
ans = x.find( 'abcd' ) ; print( ans , type( ans ) )

x = 'banana'
ans = x.find( 'ana' ) ; print( ans , type( ans ) )

x = 'banana'
ans = x.rfind( 'ana' ) ; print( ans , type( ans ) )

x = [ 1, 2 ]
x.append( [ 3, 4 ] )
ans = len( x ) ; print( ans , type( ans ) )

x = [1, 2, 3]
ans = x[ 1 : 2 ]    ; print( ans , type( ans ) )

x = 'banana'
x.replace( 'a', '' )
ans = x ; print( ans , type( ans ) )

x = [ 1, 2, 3 ]
ans = x[ x[ 1 ] ]   ; print( ans , type( ans ) )

ans = [ 11 ] + [ 12 ]   ; print( ans , type( ans ) )
ans = 1 + 1 + 1 * 2 ; print( ans , type( ans ) )
```
'A module of helper tools for CS 1112.

Contains:
   read_web - returns the contents of a webpage as a string
   distance - computes distance in miles between points on earth

Usage:

   import tools
   data = tools.read_web('http://some-link.goes/here')
   miles = tools.distance( 38.03, -78.2, 0, 0 )

import urllib.request
import math

def read_web( link ):
    '''read_link(url)
    Will go online and retrieve the contents of the given url,
    converting the answer to a string and returning it.
    '''
    stream = urllib.request.urlopen( link )
    page = stream.read()
    text = page.decode( 'UTF-8' )
    return text

def distance( lat1, long1, lat2, long2 ) :
    ''' Computes distance in miles from (lat1, long1) to (lat2, long2)
    '''
    EARTH_RADIUS = 3958.7621

    # convert latitude and longitude into radians
    degrees_to_radians = math.pi / 180.0
    phi1   = ( 90.0 - lat1 ) * degrees_to_radians
    theta1 = long1   * degrees_to_radians
    phi2   = ( 90.0 - lat2 ) * degrees_to_radians
    theta2 = long2   * degrees_to_radians

    # calculate spherical distance for unit sphere
    cos = ( math.sin( phi1 ) * math.sin( phi2 ) * math.cos( theta1 - theta2 ) +
           math.cos( phi1 ) * math.cos( phi2 ) )

    unit_sphere_distance = math.acos( cos )

    # scale distance by indicated radius
    distance = unit_sphere_distance * EARTH_RADIUS

    # got the distance, so hand it back
    return distance
Questions

Method vs Function:
Function (like len, max, print, math.sqrt) do not require an object in front of them
Method (like append, sort, capitalize, find) do require an object in front of them

Arguments: (the values inside the parentheses)
Some methods do not need arguments (because their object is enough information)
other methods and most functions do need arguments

Booleans:
they are True and False
they are created by comparator operators (< <= == > >= !=)
they can be combined with boolean operators (and or not)

s = '1,2,3,4'
i = s.find(',')
j = s.rfind(',')

print(i)
print(j)
i = s.find(',', 2)
j = s.rfind(',', 2)

print(i)
print(j)
i = s.find(',', 0, 5)
j = s.rfind(',', 0, 5)

print(i)
print(j)

where = s.find(',')
start = where == 0
print(s, 'starts with a ,:', start)

x = [1, 2, 3, 5, 7]
total = 0
for element in x:
    num = element * 2
    total = total + num
print('total:', total)

everying_so_far_is_odd = True
for element in x:
    odd = ((element % 2) == 1)
    everying_so_far_is_odd = everying_so_far_is_odd and odd
print('all odd:', everying_so_far_is_odd)

anything_so_far_is_odd = False
for element in x:
    odd = ((element % 2) == 1)
    anything_so_far_is_odd = anything_so_far_is_odd or odd
print('any odd:', anything_so_far_is_odd)
import random

print()  # pick the "random" series 1
random.seed(3.0)
print(random.randrange(0, 10))
print(random.randrange(0, 10))
print(random.randrange(0, 10))

print()  # pick the "random" series 1
random.seed(3)
print(random.randrange(0, 10))
print(random.randrange(0, 10))

print()  # pick a random "random" series
random.seed()
print(random.randrange(0, 10))
print(random.randrange(0, 10))

s = "I know everything"
emmas_list = s.split()
print(emmas_list)

answer = ''
for emmas_word in emmas_list:
    answer = answer + emmas_word + '-'

# answer = answer.strip('-')
answer = answer[ : len(answer) - 1 ] + '.'
print(answer, '.')

answer = []
for emmas_word in emmas_list:
    # or: element = [emmas_word]
    # or: element = list(emmas_word)
    element = emmas_word.split('e')
    answer.append(element)
print(answer)

whee = [1, 2, 3, 4, 5.6]

answer = ''
for number in whee:
    text = str(number)
    answer = answer + text + '
print(answer)

# What is None?
none1 = None
none2 = print()
none3 = whee.append(8)
none4 = whee.sort()
print(type(none1))
print(none3)

s = "Example text"
print(s.find("e"))
print(s.find("E"))
print(s.find('F'))
print('E' in s)
print('z' in s)

words = ['1', '2', 'three']
print(words)
words.sort()
print(words)
words.reverse()
print(words)

# result = input('two numbers, please: ')
# first, second = result.split()

for x in range(2, 14):
    print(x)

result = input('how many times? ')
count = int(result)
for i in range(count):
    print('I will ace this test')

print('every elephant'.replace('e', ''))
Purpose: discuss CSV, one of the most common data formats around

```python
import tools

def get_some_data():
    csvdata = tools.read_web('http://cs1112.ninja/datasets/csv/vastats.csv')

    # parse
    csvdata = csvdata.strip() # strip it
    lines = csvdata.split('
') # split into lines
    rows = []
    for line in lines:
        row = line.split(',')
        rows.append(row)

    # header row
    header = rows[0]
    data = rows[1:]

    # find which column we are looking for
    population_column = header.index('Total Population')

    # How do we print one column?
    for row in data:
        print(row[population_column])

    # or accumulate a list of just one column's values?
    populations = []
    for row in data:
        strpop = row[population_column]
        pop = int(strpop)
        populations.append(pop)

    print("We know of the following total populations:")
    print(populations)
    print("That totals to", sum(populations), "people")
```
Purpose: discuss CSV, one of the most common data formats around

import tools

def get_data()
    csvdata = tools.read_web('http://cs1112.ninja/datasets/csv/vastats.csv')

    # How do we parse csvdata?
    csvdata = csvdata.strip()  # strip it
    lines = csvdata.split('
')  # split into lines (\n represents a new-line character)
    # \n newline \t tab following are self: \\
    rows = []
    for line in lines:
        row = line.split(',')
        rows.append(row)

    # How do we handle that the first row is different than the others?
    header = rows[0]
    data = rows[1:]

    # do work
    look_for = '4209'
    any_have = False
    total = 0
    # How do we print the data?
    for row in data:
        has = look_for in row
        any_have = any_have or has
        num = int(row[3])
        total = total + num

    print(any_have)
    print(total)
Purpose: learn how to use tools.py to import CSV data

```
import tools

# Step 1: get the data
link = input("Type a URL to a CSV data set: ")
csvdata = tools.read_web(link)

# Step 2: parse it
csvdata = csvdata.strip() # strip it
lines = csvdata.split('\n') # split into lines
rows = []
for line in lines:
    row = line.split(',')
    rows.append(row)

# Step 3: do something
print("imported", len(rows), "rows of data")
```
import tools

data_folder = 'http://cs1112.ninja/datasets/csv/
data_ending = '.csv'

# ask the user for a dataset name, like "names" or "misspellings"
dsname = input('What is the name of the dataset? ')

# assemble the full dataset url:
# e.g., 'names' would become 'http://cs1112.ninja/datasets/csv/names.csv'
url = data_folder + dsname + data_ending
print(url)

# read that url (use tools)
contents = tools.read_web(url)

# print the contents of that dataset
print(contents)
Purpose: learn how to use tools.py to retrieve the contents of a web page

import tools

link = input('Type a URL: ')

# use tools.py to do the hard part
contents = tools.read_web(link)

# print the result
print('The page', link, 'contains')
print(contents)
Purpose: learn how to use tools.py to compute distance from Rice Hall to any location on earth

import tools

# make some constant values
rice_lat = 38.031634
rice_lon = -78.510477

latitude = input('What is your latitude? ')
longitude = input('What is your longitude? ')

other_lat = float(latitude)
other_lon = float(longitude)

# use tools.py to do the hard part
miles = tools.distance(rice_lat, rice_lon, other_lat, other_lon)

# print the result
print("That location is", miles, "miles from here")

# Your task: get the other lat/lon from the user
A module of helper tools for CS 1112.

Contains:
- `read_web` - returns the contents of a webpage as a string
- `distance` - computes distance in miles between points on earth

Usage:

```python
import tools

data = tools.read_web('http://some-link.goes/here')
miles = tools.distance(38.03, -78.2, 0, 0)
```

```python
import urllib.request
import math

def read_web(link):
    '''read_web(url)
    Will go online and retrieve the contents of the given url, converting the answer to a string and returning it.
    '''
    stream = urllib.request.urlopen(link)
    page = stream.read()
    text = page.decode('UTF-8')
    return text

def distance(lat1, long1, lat2, long2):
    '''Computes distance in miles from (lat1, long1) to (lat2, long2)
    '''
    EARTH_RADIUS = 3958.7621

    # convert latitude and longitude into radians
    degrees_to_radians = math.pi / 180.0

    phi1 = (90.0 - lat1) * degrees_to_radians
    theta1 = long1 * degrees_to_radians

    phi2 = (90.0 - lat2) * degrees_to_radians
    theta2 = long2 * degrees_to_radians

    # calculate spherical distance for unit sphere
    cos = (math.sin(phi1) * math.sin(phi2) * math.cos(theta1 - theta2) +
           math.cos(phi1) * math.cos(phi2))

    unit_sphere_distance = math.acos(cos)

    # scale distance by indicated radius
    distance = unit_sphere_distance * EARTH_RADIUS

    # got the distance, so hand it back
    return distance
```
import random

# random.seed('Stop')

print(random.choice('abcdef'))
print(random.choice('abcdef'))
print(random.choice('abcdef'))
print(random.choice('abcdef'))
print(random.choice('abcdef'))
print(random.choice('abcdef'))
Purpose: discuss CSV, one of the most common data formats around

```
csvdata = '''latitude;longitude;name;address
38.89286;-77.4272;Wendy's-Chantilly,VA ;13902 Lee Jackson Memorial Hwy, Chantilly,VA, (703) 63 1-8139
38.08089;-78.47507;Wendy's-Charlottesville,VA ;1636 Seminole Trl, Charlottesville,VA, (434) 97 3-6226
38.03377;-78.45533;Wendy's-Charlottesville,VA ;1192 Richmond Rd, Charlottesville,VA, (434) 979 -5908
37.35497;-77.41176;Wendy's-Chester,VA ;2510 W Hundred Rd, Chester,VA, (804) 796-5650
37.37366;-77.49866;Wendy's-Chesterfield,VA ;6451 Centralia Rd, Chesterfield,VA, (804) 796-9728
37.25605;-77.4112;Wendy's-Colonial Heights,VA ;1709 Boulevard, Colonial Heights,VA, (804) 526-8685
37.25543;-77.39145;Wendy's-Colonial Heights,VA ;680 Southpark Cir, Colonial Heights,VA, (804) 520-0623
38.48135;-77.99316;Wendy's-Culpeper,VA ;872 N Main St, Culpeper,VA, (540) 825-6633
'''
```

# How do we parse csvdata?

```python
csvdata = csvdata.strip()  # strip it
data = [line.strip().split(';') for line in csvdata.split('
')]  # split into lines
```

# How do we handle that some parts are numbers and some text?

# How do we handle that the first row is different than the others?

# Which Wendy's is closest to Rice 130 (38.031634, -78.510477)?
Name(s):
Computing ID(s):

Purpose: discover how to parse more complicated inputs

# Task 1: what if the delimiter is not whitespace?
ex1 = '''
Helvetical
The Private Life of Plants
Planet Earth
The King of Kong
'''
# write code that turns ex1 into
# ['Helvetical', 'The Private Life of Plants', 'Planet Earth', 'The King of Kong']
# If you modify ex1, the code should still work unchanged

ex1 = ex1.strip()
answer = ex1.split('n')
print(answer)

# Task 2: we know how to split then turn into ints; how about split and then turn into lists?
ex2 = '1,2,3;4,2,3,5,7;1,2,4,8'
# write code that turns ex2 into
# [['1', '2', '3', '4'], ['2, 3, 5, 7'], ['1, 2, 4, 8']]
# If you modify ex2, the code should still work unchanged

list_of_strings_with_commas_in_them = ex2.split(';')
print(list_of_strings_with_commas_in_them)

# Hint: this looks a LOT like the 4th example in parsing_review.py
list_of_lists_of_strings = []
for string_with_commas in list_of_strings_with_commas_in_them:
    list_of_strings = string_with_commas.split(',')
    list_of_lists_of_strings.append(list_of_strings)
print(list_of_lists_of_strings)

# Task 3: now split, then split, then intify: turn ex2 from above into
# [[1, 2, 3, 4], [2, 3, 5, 7], [1, 2, 4, 8]]

list_of_strings_with_commas_in_them = ex2.split('';')
print(list_of_strings_with_commas_in_them)

# Hint: this looks a LOT like the 4th example in parsing_review.py
list_of_lists_of_ints = []
for string_with_commas in list_of_strings_with_commas_in_them:
    list_of_strings = string_with_commas.split(',')
    list_of_numbers = []
    for string in list_of_strings:
        number = int(string)
        list_of_numbers.append(number)
    list_of_lists_of_ints.append(list_of_numbers)
print(list_of_lists_of_ints)

# if X is your result from task 3, what is min(max(x)) * max(min(x))?
Purpose: review common ways of parsing input

The following uses literal strings instead of input so we can see what’s happening more easily

```
print('"
-----------------------------------
The easiest kind: no parsing necessary, use the string
"

datum1 = 'Celery raw develops the jaw'
# no parsing necessary; go directly on to the task at hand
print('You typed', datum1.count('e'), ''e''s')

print('"
-----------------------------------
Converting str -> number
"

datum2 = '556'
datum2i = int(datum2) # parse using the built-in function int
print('You typed half of', datum2i*2)

print('"
-----------------------------------
Converting str -> list of str
"

data3 = 'banana aardvark zoology'
data3l = data3.split() # parse using the str method split
print('You typed a total of', len(data3l), 'words from', min(data3l), 'to', max(data3l))

print('"
-----------------------------------
Converting str -> list of str -> list of numbers
"

data4 = '2 3 5 7 11'
data4l = data4.split() # parse using the str method split

# we can’t turn a list of str into a list of int all in one go, so we use a loop
data4li = []
for thing in data4l:
    number = int(thing)
    data4li.append(number)
print('You typed a total of', len(data4li), 'numbers totalling', sum(data4li))
```
Purpose: review common loop usages

print('''
 이것이 looping의 일반적인 사용법
''')

for i in range(5):
    print("hi")
    # we don’t need to use i; it just keeps count of how many times to loop

print('''
_kind 1: just repeat something
''')

for julio in 'I am a loop':
    print(julio)

print('''
_kind 2a: do something with each element in a collection (a string)
''')

for thing in [3, 'yes', 11.12, None]:
    print('Got another thing...')
    print('The current thing is', thing, 'and has type', type(thing))

print('''
_kind 2b-less-repeated: same as above, but with different indentation
''')

for thing in [3, 'yes', 11.12, None]:
    print('Got another thing...')
print('The current thing is', thing, 'and has type', type(thing))

print('''
_prep for find nesting: a simple loop with an outside-the-loop variable
''')

i = 2
for j in range(5):
    print(i, j)

print('''
_nesting: a simple loop with an outside-the-loop variable, but that variable was set by another loop
''')

for i in range(3):
    for j in range(5):
        print(i, j)

print('''
 accumulator pattern (and reiteration that names do not matter)
'''

cattle = 0
for feamur in ['these', 'names', 'are', 'just', 'silly']:
    wicks = len(feamur)
    cattle = cattle + wicks
print(''
The accumulator loop is now done (notice nothing printed in the loop)
Typically we use the value after the loop finishes:
''')
print(cattle)
Determine the longest word in the line with the fewest words
Determine which word is longest

text = input('Type something: ')
words = text.split()
lengths = []
for word in words:
    lengths.append(len(word))

print(words)
print(lengths)
biggest_length = max(lengths)
spot = lengths.index(biggest_length)
print(spot)
print(words[spot])

print('The longest word had', biggest_length, 'characters in it.')
print('But we still need to figure out which word that was...')
Allows the user to type several lines of text

```python
# text = input('How many lines will you type? ')
# n = int(text)

lines = []
for i in range(1, n+1):
    line = input('Type line ' + str(i) + ' of text: ')
    # print('lines =', lines)
    lines.append(line)
    # print('lines =', lines)

print()
print('Thank you. You typed: ')
for sentence in lines:
    print(sentence)
```
Purpose: display a multiplication table. You’ll need to add format calls to make it look nice.

```python
n = input('Display a table up to what number? ')
n = int(n)

print(list(range(1, 6)))

for num1 in range(1, n+1):
    row = ''
    for num2 in range(1, n+1):
        prod = num1 * num2
        row = row + str(prod) + ' '  
    print(row)
```
Purpose: review a few odds and ends

```
print("\n", format(1112, '10'), "\n")
print("\n", format('x', '10'), "\n")
text = format(1112, '>10')
print("\n", text, "\n")
print("\n", format(1112, '<10'), "\n")
print("\n", format(1112, '^10'), "\n")
```

```
print('-'*60)
input('Press enter to continue')
```

```
s = 'FebruAry'
s.lower()
print(s)
```

```
print('-'*60)
input('Press enter to continue')
```

```
print(min("10", "2") =', min("10", "2"))
print(min(10, 2) =', min(10, 2))
print(min("D", "Z") =', min("D", "Z"))
print(min("d", "Z") =', min("d", "Z"))
print(min("one", "2") =', min("one", "2"))
print(min("Ä\222ne", "â\200¡") =', min("Ä\222ne", "â\200¡"))
```

```
sl = [ '2', '4', '8', '16', '32' ]
sl.sort()
nl = [ 2, 4, 8, 16, 32 ]
nl.sort()
print(nl)
print(sl)
```

```
print('-'*60)
input('Press enter to continue')
```

```
l = [ 'a', 'b', 'c' ]
b = True
print(l =', l)
print(l[True] =', l[b])
print(l[False] =', l[False])
```

```
list0 = [1]
list1 = [2]
list_of_lists = [list0, list1]
print('list0 =', list0)
print('list1 =', list1)
print('list_of_lists =', list_of_lists)
```
print('-'*60)
input('Press enter to continue')

list_of_lists[0].append(3)
print('list0 =', list0)
print('list1 =', list1)
print('list_of_lists =', list_of_lists)
Purpose: to demonstrate one feature of the format function

```

txt = 'CS'
num = 1112

print('-'*66)
print(txt, '|', txt, '|', txt, '|', txt, '|', txt, '|', [txt])
print(num, '|', num, '|', num, '|', num, '|', num, '|', [num])
print('-'*66)
print()

txt10 = format(txt, '10')
um10 = format(num, '10')

print('-'*66)
print(txt10, '|', txt10, '|', txt10, '|', txt10, '|', [txt10])
print(num10, '|', num10, '|', num10, '|', num10, '|', [num10])
print('-'*66)
print()

txt10r = format(txt, '>10')
um10r = format(num, '>10')

print('-'*66)
print(txt10r, '|', txt10r, '|', txt10r, '|', txt10r, '|', [txt10r])
print(num10r, '|', num10r, '|', num10r, '|', num10r, '|', [num10r])
print('-'*66)
print()

txt10l = format(txt, '<10')
um10l = format(num, '<10')

print('-'*66)
print(txt10l, '|', txt10l, '|', txt10l, '|', txt10l, '|', [txt10l])
print(num10l, '|', num10l, '|', num10l, '|', num10l, '|', [num10l])
print('-'*66)
print()

txt10c = format(txt, '^10')
um10c = format(num, '^10')

print('-'*66)
print(txt10c, '|', txt10c, '|', txt10c, '|', txt10c, '|', [txt10c])
print(num10c, '|', num10c, '|', num10c, '|', num10c, '|', [num10c])
print('-'*66)
print()

print(format('Example Missive', '^60'))
print(format('15 Feb 2016', '>60'))
print('''To whom it may concern,
The following is intended to be a table to square roots.
It has been rounded, so its values are not very precise.
'''
)
print( format('Number', '^10'), format('Root', '<7'), format('Exact?', '^10') )
```
for number in range(1, 10):
    root = number ** 0.5
    root = round(root, 5)  # <- what happens if you remove the round() function?
    exact = str((root**2) == number)  # <- what happens if you remove the str() function?
    print( format(number, '^10'), format(root, '<7'), format(exact, '^10') )
Purpose: show how to use lists to do work (using the conversion and accumulator patterns)

```python
import numbers

numbers = [1, 2, 3, 4, 5]

for number in numbers:
    total = total + number

print('The sum of all those numbers is', total)
```

`''
Purpose: explore the methods in the built-in class list
```

```python
a = []
print('a = ', a)
a.append('wisdom')
a.append('happiness')
a.append('sleep')
print('a = ', a)

a = a + ('we are hoping for it'.split())
print('a = ', a)

print('max =', max(a))
print('min =', min(a))

print("a.index('for') =", a.index('for'))

print('a.reverse() =', a.reverse())
print('a =', a)

print("a.remove('are') =", a.remove('are'))
print('a =', a)

print("a.pop(1) =", a.pop(1))
print('a =', a)

print("a.insert(1, 'were') =", a.insert(1, 'were'))
print('a =', a)
```
Purpose: explore the identity of lists

```python
a = [1, 2, 3]  # a = [1, 2, 3]
b = [4, 5, 6]
c = a          # c = [1, 2, 3]
d = c[:]

print('id(a) =', id(a))
print('id(b) =', id(b))
print('id(c) =', id(c))
print('id(d) =', id(d))

print('c =', c)
a[1] = 'A1'     # a = [1, 'A1', 3]
print('c =', c)
b[2] = 'B2'
```

```python
print(a)        # [1, 'A1', 3]
print(b)
print(c)
print(d)
```
Purpose: to demonstrate that Python does not care what you put in a list

print('''
We can put any mix of any kind of values inside a list, or nothing at all''')

# we can put any kind of value inside a list
empty_list = []
list_of_numbers = [1, 2, 3]
list_of_strings = ['queue', 'are', 's']
another_list_of_strings = 'double you ex'.split()
list_of_lists = [empty_list, list_of_numbers, list_of_strings, another_list_of_strings]
whatever = [1, 'two', 3.0, True, None, [False, []], print, str]
all_lists = [empty_list, list_of_numbers, list_of_strings, another_list_of_strings, list_of_lists, whatever]
for x in all_lists:
    print(type(x), x)

print('  using for element in collection:"
    for element in x:
        print('   ', type(element), element)

print('  using for index in range(len(collection)):"
    for index in range(len(x)):
        print('   ', index, '=>', type(x[index]), x[index])

print()
print(('*' * 40)
print()

print("I've never used this in practice, but... a list can contain itself:"))

peculiar_list = [1, 2, 3, 4]
peculiar_list[2] = peculiar_list
print("peculiar_list: ", peculiar_list)
print("peculiar_list[2]: ", peculiar_list[2])
print("peculiar_list[2][2]: ", peculiar_list[2][2])
print("peculiar_list[2][2][2]: ", peculiar_list[2][2][2])

print()
print("Two lists can also each contain the other...")
nonsense1 = ['so', 'very', 'confusing;', 'do NOT do it']
nonsense2 = ['this', 'is', nonsense1]
nonsense1[1] = nonsense2
print(nonsense1)
print(nonsense2)
print(nonsense2[2])
print(nonsense2[2][1])
print(nonsense2[2][1][2])
print(nonsense2[2][1][2][1])
Purpose: use strings, lists, and for loops to capitalize all words in a string

```python
# example:
Type a sentence: I hope this works
I Hope This Works
```
Purpose: explore three basic kinds of sequences

```
print(''

str is a sequence type: given a str s, you can use len(s), s[i], and for c in s ''

s = 'abcdef'
print('s =', s)
print('s[2] =', s[2])
print('len(s) =', len(s))
print('s[3:5] =', s[3:5])
for character in s:
    print('character', character)

print(''

list is a sequence type: given a list l, you can use len(l), l[i], and for e in l ''

s = 'I love to use lists'
l = s.split() # split creates a list
print('l =', l)
print('l[2] =', l[2])
print('len(l) =', len(l))
print('l[3:5] =', l[3:5])
for element in l:
    print('element', element)

print(''

range is a sequence type: given a range r, you can use len(r), r[i], and for e in r ''

r = range(3, 12)
print('r =', r)
print('r[2] =', r[2])
print('len(r) =', len(r))
print('r[3:5] =', r[3:5])
for element in r:
    print('element', element)

print(''
There are three ways to make a range:
    range(n) contains [0, 1, 2, ... n-1]
    range(a, b) contains [a, a+1, a+2 ... b-1]
    range(a, b, c) contains [a, a+c, a+2*c, ... largest a+k*c that is smaller than b]
''

print('range(1,12,2) contains', list(range(1, 12, 2)) )
print('range(1,12,3) contains', list(range(1, 12, 3)) )
```
print('range(1,12,4) contains', list(range(1, 12, 4)))
Purpose: demonstrate two ways to do the same thing

print('"

----------------------------------------------------
Approach 1: using [] and literal values as indices
"
)
s = 'abcdefg'
x = s[0]
print('letter', x.upper())
x = s[1]
print('letter', x.upper())
x = s[2]
print('letter', x.upper())
x = s[3]
print('letter', x.upper())
x = s[4]
print('letter', x.upper())
x = s[5]
print('letter', x.upper())

print('"

----------------------------------------------------
Approach 2: using the for keyword to make a for-loop
"
)
for julio in "I don't have anything to type".split():
    print('letter', julio.upper())
import random

# see http://www.cs.virginia.edu/~cs1112/documents/modules/random/ for a
# discussion of interval notation

# important one below is \( x, y \) -- this means all values starting with \( x \)
# and up to but not including \( y \)

# if we do not set the seed, the random number sequence differs with every run

x = random.random()  # random returns a float number from
# float interval \([0, 1)\)

# because we did not set the seed, every time we run
# the program the values assigned to \( x \) is different

print( "x = ", x )

print()

# if we pass a seed value, the random number generator is configured to
# produce the same sequence of random values from that point on.

random.seed( 1112 )

y = random.random()  # first number from seed(1112) is always
# 0.2814347004335006. how do we know this, we ran the
# program and recorded the value

print( "y = ", y )

print()

# rand.uniform( x, y ) returns a random float from the interval \([ x, y)\)

n1_uniform = random.uniform( 1.5, 7.25 )

random.seed()  # # pick a new sequence of numbers based on the current time

n2_uniform = random.uniform( 1.5, 7.25 )

print( "uniform values: ", n1_uniform, "and", n2_uniform )

print()

# note: random.random() and random.uniform( 0, 1 ) are synonyms

# rand.gauss( m, s ) returns a random normally distributed float
# the generated value have mean \( m \) and standard deviation \( s \)

n1_gauss = random.gauss( 1.5, 1.25 )  # mean = 1.5, std dev = 1.25
n2_gauss = random.gauss( 1.5, 1.25 )

print( "gaussian values: ", n1_gauss, "and", n2_gauss )

print()

bottom = 2
top = 7
step = 2

# random.randrange( x ) returns a random integer value from the interval
# [ 0, x ); values are equally likely to occur;
# we say that the generated numbers are from base x

x = random.randrange( top )

# random.randrange( x, y ) returns a random integer value from the interval
# [ x, y ); values are equally likely to occur;

print( "randrange( ",top," ) gave", x )

print()

x = random.randrange( bottom, top )
print( "randrange( ", bottom, ",", top," ) gave", x )

print()

# random.randrange( x, y, s ) returns a random integer value from the
# interval [ x, y ); values are equally likely to occur; generated numbers are of form
# x + ( i * s )

x = random.randrange( bottom, top, step )
print( "randrange( ", bottom, ",", top, ",", step, " ) gave", x )

print()

# random.choice( s ) if s is a string, the method returns a random
# character from the string

alphabet = "qwertyuiopasdfghjklzxcvbnm"
letter = random.choice( alphabet )
print( "We like letter", letter )

print()

# random.choice( s ) if s is a sequence, the method returns a random
# element from the sequence

days = [ 'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday' ]
appointment = random.choice( days )
print( "We will next see you on", appointment )

print()

# random.shuffle( s ) if s is a sequence, randomizes the ordering of
# the values in the sequence

random.shuffle( days )
print( "The ordering of days is now", days )

print()
```python
# str is built-in Python class type. no import is needed
#
# len( s ): remember we already saw the built-in function len().
# function returns the number of characters making up s
#
# print( "len( ‘hi’ ) =", len( ‘hi’ ) )
#
# str( n ): remember we already saw the built-in function str().
# the function produces a new str that is a textual representation
# of n
#
# s = str( 1.2 )
#
# print( "str( 1.2 ) =", s, "which has type", type( s ) )
# print( "1.2 =", 1.2, "which has type", type( 1.2 ) )
#
# alphabet = "abcdefghijklmnopqrstuvwxyz"
#
# strings can be sliced. every slicing produces a new str
#
# s[ i : j ]: returns a substring of s composed out of the
# characters at indices i to j-1; that is the characters
# at indices interval [ i, j )
#
# remember indices start counting from 0
#
# sliced = alphabet[ 13 : 16 ]
# print( "sliced =", sliced )
#
# s[ : j ]: returns a substring of s composed out of the
# characters at indices 0 to j-1; that is the characters
# at indices interval [ 0, j )
#
# front = alphabet[ : 4 ] # starts counting at 0; uses [0, ) interval
# print( "front =", front )
#
# s[ i : ]: returns a substring of s composed out of the
# characters at indices i to len(s ) - 1; that is the characters
# at indices interval [ i, len( s ) )
#
# back = alphabet[ 20 : ]
# print( "back =", back )
#
# s[ : ]: returns a new copy of s; i.e., the characters at indices
# i to len( s ) - 1; that is the characters at indices interval [ 0, len( s ) )
#
# everything = alphabet[ : ]
# print( "everything =", everything )
#
```
```python
# s[i]: returns a new unit-length substring of s; i.e., the
# character at index i; that is, the characters at indices interval
# [i, i+1)

letter = alphabet[2]
print("letter =", letter)

print(alphabet)
capital = alphabet.capitalize()
print(capital)
all_caps = alphabet.upper()
print(all_caps)
capital_d = alphabet.capitalize(3)
print(capital_d)

fourth = alphabet[3]
fourth_capital = fourth.upper()
first_part = alphabet[:3]
last_part = alphabet[4:]
print(fourth_capital)
print(first_part)
print(last_part)
capital_d = first_part + fourth_capital + last_part
print(capital_d)

print(alphabet.split)
split_on_d = alphabet.split('stu')
print(split_on_d)
print(type(split_on_d))

e1, e2 = split_on_d
print(e1)
print(e2)

text = input("Type some text: ")

# print the number of vowels (aeiouAEIOU) they typed
small_text = text.lower()
a = small_text.count('a')
e = small_text.count('e')
i = small_text.count('i')
o = small_text.count('o')
u = small_text.count('u')

print(a + e + i)
```
# purpose: determine the worth of a piggy bank

# problem statement:
# Prompts its user for four values: a number of quarters, dimes, nickels, and pennies. Then computes and displays how much the indicated coins are worth.

print()

# prompt and get the various amounts of change

print()

# convert input into integer values

# calculate total worth

# display worth
# expand input processing capabilities by splitting
# input string into component words

# get input

data = input('What are your three favorite words: ')

print()

# determine individual words w1, w2, and w3

# display results
# introduce relational operators

# get two numbers from user

print()

sa = input('Enter number: ')
sb = input('Enter number: ')

print()

# convert inputs to integer

a = int(sa)
b = int(sb)

# compare and record results

b1 = (a < b)  # < is less than
b2 = (a > b)  # > is greater than
b3 = (a <= b) # <= is less than or equal to
b4 = (a >= b) # >= is greater than or equal to
b5 = (a == b) # == is equals
b6 = (a != b) # != is not equals

# display results

print(a, '<', b, ':', b1)
print(a, '>', b, ':', b2)

print()

print(a, '<=', b, ':', b3)
print(a, '>=', b, ':', b4)

print()

print(a, '==', b, ':', b5)
print(a, '!=', b, ':', b6)

print()
Purpose: expose you to the idea that True and False are values

print('What is 12?')
print('\tA', type(12) )

print('What operators do we know for ints?')
print('\t+   -   *   /   %   //   %')

print('If I apply an operator to ints what do I get back?')
print('\tA', type(12 + 13) )

print()

print('What is "12"?')
print('\tA', type("12") )

print('What operators do we know for strs?')
print('\t+   [i]   [i:j]')

print('If I apply an operator to strs what do I get back?')
print('\tA', type('1112'[1:3]) )

print()

print('What is True?')
print('\tA', type(True) )

print('What operators do we know for bools?')
print('\tand    or    not')

print('If I apply an operator to bools what do I get back?')
print('\tA', type(True and False) )

print()

print('There are only two bool-type values: True and False')

print()

print( False, 'and', False, 'is', False and False )
print( False, 'and', True, 'is', False and True )
print( True, 'and', False, 'is', True and False )
print( True, 'and', True, 'is', True and True )

print()

print( False, 'or', False, 'is', False or False )
print( False, 'or', True, 'is', False or True )
print( True, 'or', False, 'is', True or False )
print( True, 'or', True, 'is', True or True )

print()

print( 'not', False, 'is', not False )
print( 'not', True, 'is', not True )
# swapping values

# get and echo inputs

w1 = input('Enter a word: ')
w2 = input('Enter another word: ')

print()

print('before swapping')

print('w1 =', w1)
print('w2 =', w2)

print()

# swap the values of w1 and w2

# print results

print('After swapping')

print('w1 =', w1)
print('w2 =', w2)
# a cautionary tale

# get Celsius temperature

c = int( input('Enter Celsius temperature: ' ) )

print()

# compute Fahrenheit equivalents of c; for f1 use /, for f2 use //

# display results

print( c, 'C =', f1, 'F' )
print()
print( c, 'C =', f2, 'F' )
# processing numeric input

```
import math

print()
```

# get inputs

```
r = input( 'Enter an integer radius: ' )
d = input( 'Enter a decimal: ' )
```

# compute c - the length of circle with radius r

```
# compute i - the multiplicative inverse of d
```

# display results

```
print( c )
print( i )
```
# interactive - program gets data from user and produces info

# get input
s = input('Enter a string: ')

############################################################################
# determine n - the length of s
n = ...

# display result
print()
print('The input has length: ', n)

############################################################################
# determine last_index - the index of the last character in s
last_index = ...

# determine last_character - the last character in s
last_character = ...

# display result
print()
print('The last character (at index', last_index, ') is', last_character)

############################################################################
# determine one_third_n - the index of a character about 1/3 of the way through s
one_third_n = ...

# determine two-thirds_n - the index of a character about 2/3 of the way through s
two_thirds_n = ...

# determine middle - the middle third of s
middle = ...

# display result
print()
print('The middle of', s, 'is', middle)
Purpose: to explore the following ideas:

A variable indicates or contains an "object" (also called a value)

Each object
- has a "type" or "class", such as int, str, or float
- has an "identity", "id", or "address" which uniquely identifies that particular object
- has "content" or "value" which is what we see when we look at it

```python
x = 11
print(f'Variable x indicates object #\'{id(x)}')
print(f'  object #\'{id(x)}'+'\'s type is', type(x))
print(f'  object #\'{id(x)}'+'\'s value is', x)
print()
print(f'We often refer to variables as if they were the objects they indicate:')
print(f'  x is', x)
print(f'  x\'s value is', x)
print(f'  x is an', type(x).__name__)
print(f'  x\'s type is', type(x).__name__)
print(f'We also sometimes prefix the variable or value by its type:')
print(f'  ', type(x).__name__, 'variable x')
print(f'  ', type(x).__name__, 'value', x)
print()
print('x can indicate different objects at different times:')
x = 11
print(f'  variable x now indicates object #\'{id(x)}')
x = 12
print(f'  variable x now indicates object #\'{id(x)}')
print(f'If a variable *never* changes value, we call it a "constant"')
print()
print('Multiple objects can have the same value:')
y = '1112'
z = str(1112)
print(f'  object #\'{id(y)}'+'\'s value is', y)
print(f'  object #\'{id(z)}'+'\'s value is', z)
print()
print('Multiple variables can indicate the same object:')
x = y
print(f'  variable x indicates object #\'{id(x)}')
print(f'  variable y indicates object #\'{id(y)}')
print(f'  variable z indicates object #\'{id(z)}')
print(f'Variables that indicate the same object are called "aliases" of one another.')
```
# using variables

print( )

# first generation there are 2 rabbits. each successive
# generation doubles the number of rabbits

nbr_rabbits = 2
generation = 1

print( 'Generation: ', generation )
print( 'rabbits:    ', nbr_rabbits )
print( )

# next generation

nbr_rabbits = 4
generation = 2

print( 'Generation: ', generation )
print( 'rabbits:    ', nbr_rabbits )
print( )

# next generation

nbr_rabbits = 8
generation = 3

# next generation

print( 'Generation: ', generation )
print( 'rabbits:    ', nbr_rabbits )
print( )

nbr_rabbits = 16
generation = 4

# next generation

print( 'Generation: ', generation )
print( 'rabbits:    ', nbr_rabbits )
print( )
introduce string manipulation

print()
print('abc' 'def')
print('abc' + 'def')
print()
# print( 'C' + 13 + ' H' + 20 + ' O' )
print( 'C' + str(13) + ' H' + str(20) + ' O' )
print('wahoo wah ' * 3)
print()
print('abcdefghij')
print()
print('abcdefghij'[0])
print('abcdefghij'[4])
print('abcdefghij'[9])
print()
#    123    [,)
#    0123456789
print('abcdefghij'[1:4])
print()
print(len('abcdefghij'))
print()
print(len('a b c d'))
print()
# print( 'you’re' )
print('you\'re')
print()
print('abcd\nefgh\nij')
print()
print(len('abcd\nefgh\nij'))
print()
print(len('abcd\ntefgh\ntij'))
print()
print("you’re")

print(""
    "I’m too busy mopping the floor to turn off the faucet."
    --- Anonymous
"")
# reminder that Python arithmetic manipulates values with finite precision

```
print()
print( 1000000000000.0 + 0.00001 )
print( 1000000000000.0 + 0.00001 - 1000000000000.0 )
print( 0.1 + 0.1 + 0.1 )
```
# Name(s):
# Computing ID(s):

# For each of the following, have Python do the math (e.g., write 8+9+10 not 27)
# We encourage using variables, but it is not required

import math

# Find the sum of 8, 9, and 10 (we did this one for you)
print('the sum of 8, 9, and 10 is', 8 + 9 + 10)

# Find the product of 8, 9, and 10

# Compute the number of seconds in a year (365 days, 24 hours/day, 60 minutes/hour, etc)

# Compute the number of inches in a one mile (hint: there are 5280 feet per mile)

# Compute the number of 2ft square tiles to cover the floor of a 10ft by 12ft room

# Find the average age of five people around you

# Find the volume of a sphere with radius 1 (forgot the formula? Wikipedia knows all...)
volume = (4/3)*(math.pi)*(1**3)

# 1 light year is 5.87849981 \times 10^{12} miles
light_year_in_miles = 5.87849981 * (10**12)
# The Andromeda galaxy is 2.9 million light years away. How many miles away is it?

# How many years would it take to get to the Andromeda galaxy at 65mph?
speed = 65 # mph
distance_in_light_years = 2.9 * (10**6)
distance = distance_in_light_years * light_year_in_miles # miles
hours = distance / speed
hours_in_a_year = 365 * 24
years = hours / hours_in_a_year
print("It would take", years, "years to drive to Andromeda")

# The universe is 15 billion years old, how many seconds old is it?
answer = 15 * (10**9) * 365 * 24 * 60 * 60
print("The universe is", answer, "seconds old")
# purpose: demonstrate arithmetic

# name some values
a = 4
b = 10
c = -5
d = 100

x = 4.25
y = 10.75
z = 2.50
w = -3.14

print()

# display some calculations with integers

print( 'Integer manipulation' )

print()

print( a, '+', b, '=', a + b )
print( b, '-', a, '=', b - a )
print( a, '*', b, '=', a * b )
print( b, '/', a, '=', b / a )

print()

print( b, '//', a, '=', b // a ) # integer division
print( b, '%', a, '=', b % a ) # remainder or mod
print( b, '**', d, '=', b ** d ) # exponentiation

print()

print( '-', b, '=', -b )

print()

print( 'abs(', c, ') =', abs( c ) )
print( 'int(', y, ') =', int( y ) )

print()

# display some calculations with floats

print( 'Float manipulation' )

print()

print( x, '+', y, '=', x + y )
print( y, '-', x, '=', y - x )
print( x, '*', y, '=', x * y )
print( y, '/', x, '=', y / x )

print()

print( y, '//', x, '=', y // x )
print( y, '%', x, '=', y % x )
print( x, '**', z, '=', x ** z )
print() 
print( -y, '=' , -y ) 
print() 
print( 'abs( w ) =', abs( w ) ) 
print( 'float( a ) =', float( a ) ) 
print() 
# display some calculations with mixed operands 
print( 'Mixed numeric manipulation' ) 
print() 
print( a, '+', y, '=' , a + y ) 
print( y, '-', b, '=' , y - b ) 
print( a, '*', y, '=' , a * y ) 
print( y, '/', b, '=' , y / b ) 
print() 
print( y, '//', b, '=' , y // b ) 
print( y, '%', b, '=' , y % b ) 
print( a, '**', z, '=' , a ** z ) 
print()
# why guidelines

# adapted from:
#     http://google-styleguide.googlecode.com/svn/trunk/pyguide.html

print()

print(‘The point of having style guidelines is to have a common vocabulary ’)
print(‘of coding so people can concentrate on what you are saying rather ’)
print(‘than on how you are saying it.’)

print()
''' Purpose: naming allows abstraction.
    Authors: Jim Cohoon and Luther Tychonievich
    IDs:     jpc and lat7h
'''

import math
# import is a keyword
# math is a module (or library) that we want to be able to use

print()
# . means 's -- it selects something from something else (math's pi)
print( math.pi )
print( math.e )
print( 'math.pi' )
# print( math.Euler ) # semantic error -- ok syntax, but not possible to run

# identifiers in this file: math, print, pi, e
# keywords in this file: import
# literals in this file: 'math.pi', 'math.pi ='

# we want a line whose output is
# math.pi = 3.141592653589793

print( 'math.pi =', math.pi )
# authors: Jim Cohoon and Luther Tycho nievich
# ids:     jpc and lat7h
# purpose: illustrate printing in Python

print( 'We learn best,' + ' when we must invent.' )
print( "--Jean Piaget" )

print() # makes a blank line (can make the output easier to read)
# print( We learn best, + when we must invent. ) # does not work without quotes
# anything following a ‘#’ is a comment, and is ignored by Python

# print(...) is a built-in function
# functions (like print) must have parentheses following them
# things inside the parentheses are the "arguments" to print (what to print)

# 'We learn best, when we must invent. -- Jean Piaget' is text
# Python calls text "string"s
# Strings represented literally with ' ' or " " are called "string literals"
# both kinds ('' and "") are OK; generally '' is preferred

# spaces, newlines, etc, not in string literals are called "white space"
# a semicolon (;) means "this is the end of the line", not needed in python

# 2 + 3 is 5         (addition)
# "2" + "3" is "23"  (concatentation)

print( 11, '+', 12, '=', 11 + 12 )
print( '11', '+', '12', '=', '11' + '12' )
# print( 11, +, 12, =, 11 + 12 ) # will not work; can’t print a + without quotes
print('I think there is a world market for five computers')