Harolds notes on Lists.

Append vs. Concatenate.

l1 = [1,2,3]

l2 = [4,5,6]

l3c = l1 + l2

print(l3c)

# concatenation combines the elements of two lists

# Using concatenation, the result (l3c) was a new list containing all of the members of the two lists

* **[1, 2, 3, 4, 5, 6]**

l1.append(l2)

# append adds l2 to the end of the list l1 as nested list

# The second list became a member of the first list when the second was appended to the first.

print(l1)

* **[1, 2, 3, [4, 5, 6]]**

Can’t use len() to access elements of a nested list.

my\_list = [[1, 2, 3, 4], [4, 5, 6, 7], [7, 8, 9, 10], [10, 11, 12, 13]]

nested\_list = my\_list[0]

print(len(nested\_list))

* TypeError: object of type ‘int’ has no len()

This is because when you access an element of a list using an index, you get a single element of the list, which could be of any data type, including another list. You CAN access an element of that list. Just not the len().

This is okay.

my\_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

nested\_list = my\_list[1]

length = len(nested\_list)

print(length) # Output: 3

[[1,2,3],[4,5,6],[7],[8,9,10,11]]

def undup(l):

# Create a new list

new\_list = []

for item in l:

# Check if item is not in the new list

if item not in new\_list:

new\_list.append(item)

return new\_list

s = "Mary had a little lamb"

#works with strings

print(undup([1, 2, 3, 3, 4, 1, 5, 5, 5]))

print(undup(s))

Write a function numres(nl). It's single parameter is a list of numbers. It returns a list of numbers. The first item in the output list is the count of numbers in the input. The second number is the sum. The third number is the mean.

Good **Better**

|  |  |
| --- | --- |
| def numres(nl):  sum = 0  for i in nl:  sum = sum + i  mean = sum/len(nl)  return [len(nl),sum, mean]  t = [1, 2, 3, 3, 4, 1, 5, 5, 5]  print(numres(t)) | def numres(nl):  count = len(nl)  total = sum(nl)  if count == 0:  mean = 0  else:  mean = total / count  return [count, total, mean] |

a,b,c = [1, ['a', 'b', ['x', 'y']]]

won’t work, but this will

xlist = [1, ["a", "b", ["x", "y"]]]

a, (b, c, (d, e)) = xlist

This gets you acces to each element of xlist, but variable. You could access it by iteration, but that requires recursion. Not yet there.

[67, 28, 45, 28, 61, 49, 31, 18, 97, 89, 25, 11, 62, 32, 92, 85, 2, 12, 34, 12]

import random

a = []

for i in range(20):

a.append(random.randint(1, 100))

sorted\_list = a.sort()

print(a)

* [1, 1, 16, 17, 20, 22, 40, 44, 49, 52, 58, 66, 75, 79, 84, 84, 84, 92, 94, 100]

print(sorted\_list)

=> None

**# no sorting occurs if .sort() is used on right hand side**

bigger = a.append(150)

print(bigger)

* None

**# append doesn’t work on right side when assigned to variable, but works on it’s own.**

print(a)

* [1, 1, 16, 17, 20, 22, 40, 44, 49, 52, 58, 66, 75, 79, 84, 84, 84, 92, 94, 100, 150]

Create a function how\_many\_a\_in\_li with two parameters a and li. The parameter a is a single object of any type. The parameter li is a list. The function returns the number of times the object a is found in the list li. Test your function by setting li equal to [1,2,2,3,2,4] and a to 2.

def how\_many\_a\_in\_li(a, li):

return li.count(i)

li = [1, 2, 2, 3, 2, 4]

print(how\_many\_a\_in\_li(2, li))

* 3

def how\_many\_a\_in\_li(a,li):

count =0

for i in li:

if i ==a:

count+=1

return count

li = [1, 2, 2, 3, 2, 4]

print(how\_many\_a\_in\_li(2, li))

* 3

or use:

Test the function again using the string "Mary had a little lamb" as the second argument. Count the number of times the letter 'a' appeared in this string. Does it matter that the second argument is a string, not a list?

Both work with string

li = "Mary had a little lamb"

print(how\_many\_a\_in\_li("a", li))

* 4

**No. The only restrictions on argument types are imposed by the usage of the parameter within the body of the function.**

Use a for loop to create the following list of tuples.

[(1,1),(2,4),...,(10,100)]

li = [] # creating an empty list

for num in range(1, 10):

li.append((num, num\*\*2)) # increasing the list with tuples (number, square of number)

print(li) # printing the created list

repeat Task 1 with a list comprehension

list\_of\_tuples = [(x, x\*\*2) for x in range(5)]

print(list\_of\_tuples)

* [(0, 0), (1, 1), (2, 4), (3, 9), (4, 16)]

**Dictionary**

weights\_dict = {

'Joe': 175,

'Tom': 190,

' Dick': 150

}

print(weights\_dict)

* {'Joe': 175, 'Tom': 190, ' Dick': 150}

Add Harry, who weighs 180 pounds to the dictionary and print the new dictionary.

weights\_dict ["harry"] = 180.

Remove Joe from the dictionary and print the new dictionary.

del adict["Joe"]

print(adict)

{'Tom': 190, 'Dick': 150, 'harry': 180.0}

Use the dict to look up Dick’s weight and print it.

print(adict["Dick"])

* 150

Iterate over the items in the dictionary and calculate the sum of the weights. Print the sum.

sum = 0 set empty variable to 0

for I in list.values()

sum += i

for x in adict.values():

sum += x

print(sum)

520.0

Use the method items() to get the items in adict as returned\_items. Print the object returned by the items() method. What is the type of this object?

Use the method items() to get the items in adict as returned\_items. Print the object returned by the items() method. What is the type of this object?

returned\_items **=** adict**.**items()

print(returned\_items)

print(type(returned\_items))

dict\_items([('Tom', 190), ('Dick', 150), ('harry', 180.0)])

<class 'dict\_items'>

Can you iterate over returned\_items?

**for** i **in** returned\_items:

print(i)

print(type(i))

('Tom', 190)

<class 'tuple'>

('Dick', 150)

<class 'tuple'>

('harry', 180.0)

<class 'tuple'>

Can you use an integer index to obtain one of the things in returned\_items?

No

Could you convert the object to a list and use an integer index. How about sorting.

print(list(returned\_items)[0])

print(sorted(returned\_items)[0])

new\_dict = {}

for key, value in sorted(adict.items()):

new\_dict[value] = key

print(new\_dict)

{150: 'Dick', 190: 'Tom', 180.0: 'harry'}

What happens if we do this and the same value occurs twice. Add Jerry with a weight of 150 and run the process again