Complete the script so that it removes duplicate items from the list a .

Answer 1:

a = ["1", 1, "1", 2]

a = list(set(a))

print(a)

Explanation 1:

We used a set function to convert the list to a set that would intermediately produce {'1', 1, 2} with no duplicates because set objects cannot contain duplicates. Then using list we converted the set back to a list. The drawback here is that the original order of the items is lost. If you need to preserve the order, you may want to use the solution in Answer 2 below.

Answer 2:

from collections import OrderedDict

a = ["1", 1, "1", 2]

a = list(OrderedDict.fromkeys(a))

print(a)

Explanation 2:

Ordered dictionaries are another data type in Python that, unlike sets and normal dictionaries they preserve the order of the items. Here OrderedDict.fromkeys(a) would produce a OrderedDict like [('1', None), (1, None), (2, None)] . Then we would convert that OrderedDict to a list.

Answer 3:

a = ["1", 1, "1", 2]

b = []

for i in a:

if i not in b:

b.append(i)

Explanation 3:

This is another solution that would preserve the original order. We go through all the original list items and append them to the new list if they have not been appended before. The downside of this is that the operation can take a lot of time if the list as large as we need to access both lists and perform a conditional in each iteration.

Create a dictionary that contains the keys a and b and their respective values 1 and 2 .

Answer 1:

d = {"a": 1, "b": 2}

Explanation 1:

Using curly brackets is one way to create a dictionary.

Answer 2:

d = dict(a = 1, b = 2)

Explanation 2:

A dict function is another way to create a dictionary. dict is also used to convert other objects to a dictionary.

Please complete the script so that it prints out the value of key b .

d = {"a": 1, "b": 2]

Answer:

d = {"a": 1, "b": 2}

print(d["b"])

Explanation:

As you see, accessing dictionary values follows the same syntax as accessing list items. The difference is that lists have indexes, while dictionaries have keys that you create by yourself.

Calculate the sum of the values of keys a and b .

d = {"a": 1, "b": 2, "c": 3}

Answer:

d = {"a": 1, "b": 2, "c": 3}

print(d["b"] + d["a"])

Explanation:

It's as easy as that. However, if you want to do the sum of all dictionary values, you need to take another approach instead of accessing all values one by one. We're going through that approach later on in another exercise.

Add a new pair of key (e.g. c ) and value (e.g. 3 ) to the dictionary and print out the new dictionary.

d = {"a": 1, "b": 2}

Answer:

d = {"a": 1, "b": 2}

d["c"] = 3

print(d)

Explanation:

Adding pairs of keys and values is straightforward, as you can see. Note, though, that you cannot fix the order of the dictionary items. Dictionaries are unordered collections of items.

calculate the sum of all dictionary values.

d = {"a": 1, "b": 2, "c": 3}

Answer:

d = {"a": 1, "b": 2, "c": 3}

print(sum(d.values()))

Explanation:

d.values() returns a list-like dict\_values object while the sum function calculates the sum of the dict\_values items.

Exercise for reference:

Filter the dictionary by removing all items with a value of greater than 1.

d = {"a": 1, "b": 2, "c": 3}

Answer:

d = {"a": 1, "b": 2, "c": 3}

d = dict((key, value) for key, value in d.items() if value <= 1)

print(d)

Explanation:

Here we're using a dictionary comprehension. The comprehension is the expression inside dict() . The comprehension iterates through the existing dictionary items, and if an item is less or equal to 1, the item is added to a new dict. This new dict is assigned to the existing variable d , so we end up with a filtered dictionary in d.

Create a dictionary of keys a, b, c where each key has as value a list from 1 to 10, 11 to 20, and 21 to 30, respectively. Then print out the dictionary in a nice format.

Answer:

from pprint import pprint

d = {"a":list(range(1, 11)), "b":list(range(11, 21)), "c":list(range(21, 31))}

pprint(d)

Explanation:

We're using ranges here to construct the lists. We're also using the built-in Python pprint module, which is used to print out well-formatted views of datatypes in Python.

Question: Access the third value of key b from the dictionary.

from pprint import pprint

d = dict(a = list(range(1, 11)), b = list(range(11, 21)), c = list(range(21, 31)))

Expected output:

13

Ans:pprint(d[‘b’][2])

Question: Please complete the script so that it prints out the expected output.

d = dict(a = list(range(1, 11)), b = list(range(11, 21)), c = list(range(21, 31)))

Expected output:

b has value [11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

c has value [21, 22, 23, 24, 25, 26, 27, 28, 29, 30]

a has value [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

for (keys,value in d.items()):

print(keys, “ has values” ,value)

Make a script that prints out letters of the English alphabet from a to z, one letter per line in the terminal.

Answer:

import string

for letter in string.ascii\_lowercase:

print(letter)

Explanation:

string is a built-in module and string.ascii\_lowercase returns a string object containing all 26 letters of the English alphabet. Then we simply iterate through that string and print out the string items.

Make a script that prints out numbers from 1 to 10.

Answer:

for i in range(1,11):

print(i)

Explanation:

A for loop is used to repeat an action (i.e. print ) until the iterating sequence (i.e. range ) is consumed. In our case, it would print all items of the range one by one.

Exercise for reference:

Create a function that calculates acceleration.

Answer (Python 3):

def acceleration(v1, v2, t1, t2):

a = (v2 - v1) / (t2 - t1)

return a

print(acceleration(0,10,0,20))

Explanation (Python 3):

The first three lines are where we create the function. A function definition is like a like a blueprint. Then in the last line, we're printing out the function output. The output is whatever is returned by the return statement.

Answer (Python 2):

def acceleration(v1, v2, t1, t2):

a = float(v2 - v1) / float(t2 - t1)

return a

print(acceleration(0,10,0,20))

Explanation (Python 2):

If you were creating this in Python 2, the solution would need to have two float functions converting the two differences to float numbers because if the differences are integers, Python will also output an integer (e.g., 3 / 2 outputs 0). In Python 3, you don't have to convert to floats.

Why is there an error in the code, and how would you fix it?

def foo(a, b):

print(a + b)

x = foo(2, 3) \* 10

Answer:

Line 4 throws a TypeError because Python cannot multiply a None type object with an integer. The function output is what produces a None object because the function definition is not returning anything. Fix it by using return instead of print :

def foo(a, b):

return a + b

x = foo(2, 3) \* 10