```
nested1 = [['a', 'b', 'c'],['d', 'e'],['f', 'g', 'h']]
print(nested1[0])
print(len(nested1))
nested1.append(['i'])
print("-------")
for L in nested1:
    print(L)
```

```
['a', 'b', 'c']
3
-------
['a', 'b', 'c']
['d', 'e']
['f', 'g', 'h']
['i']
```

```
nested1 = [['a', 'b', 'c'],['d', 'e'],['f', 'g', 'h']]
y = nested1[1]
print(y)
print(y[0])

print([10, 20, 30][1])
print(nested1[1][0])
```

```
['d', 'e']
d
20
d
```

```
nested1 = [['a', 'b', 'c'],['d', 'e'],['f', 'g', 'h'], ['i']]
nested1[1] = [1, 2, 3]
nested1[1][0] = 100

nested2 = [{'a': 1, 'b': 3}, {'a': 5, 'c': 90, 5: 50}, {'b': 3, 'c':
"yes"}]

def square(x):
    return x*x

L = [square, abs, lambda x: x+1]

print("****names****")
for f in L:
    print(f)

print("****call each of them****")
for f in L:
    print(f(-2))

print("****just the first one in the list****")
```

```
print(L[0])
print(L[0](3))
```

```
****names****
<function square at 0x7fc98ff568e0>
<built-in function abs>
<function <lambda> at 0x7fc98ff56840>
****call each of them****
4
2
-1
****just the first one in the list****
<function square at 0x7fc98ff568e0>
9
```

```
animals = [['cat', 'dog', 'mouse'], ['horse', 'cow', 'goat'],
['cheetah', 'giraffe', 'rhino']]
idx1=animals[1][0]
print(idx1)
```

```
horse
```

```
data = ['bagel', 'cream cheese', 'breakfast', 'grits', 'eggs',
'bacon', [34, 9, 73, []], [['willow', 'birch', 'elm'], 'apple',
'peach', 'cherry']]
plant=data[7][0][0]
print(plant)
```

```
willow
```

```
info = {'personal_data':
         {'name': 'Lauren',
          'age': 20,
          'major': 'Information Science',
          'physical_features':
             {'color': {'eye': 'blue',
                        'hair': 'brown'},
              'height': "5'8"}
         },
       'other':
         {'favorite_colors': ['purple', 'green', 'blue'],
          'interested_in': ['social media', 'intellectual property',
'copyright', 'music', 'books']
         }
      }
color=info['personal_data']['physical_features']['color']
print(color)
```

```
{'eye': 'blue', 'hair': 'brown'}
```

```
d = {'key1': {'a': 5, 'c': 90, 5: 50}, 'key2':{'b': 3, 'c': "yes"}}
print(d)
d[5] = {1: 2, 3: 4}
print(d)
d['key1']['d'] = d['key2']
print(d)
```

```
{'key1': {'a': 5, 'c': 90, 5: 50}, 'key2': {'b': 3, 'c': 'yes'}}
{'key1': {'a': 5, 'c': 90, 5: 50}, 'key2': {'b': 3, 'c': 'yes'}, 5:
{1: 2, 3: 4}}
{'key1': {'a': 5, 'c': 90, 5: 50, 'd': {'b': 3, 'c': 'yes'}}, 'key2':
{'b': 3, 'c': 'yes'}, 5: {1: 2, 3: 4}}
```

Loads takes a string as input and returns a dictionary; dumps take a dictionary as input and returns a string.

```
import json
a_string = '\n\n\n{\n "resultCount":25,\n "results": [\
n{"wrapperType":"track", "kind":"podcast", "collectionId":10892}]}'
print(a_string) #\n is simply doing line breaks
d = json.loads(a_string)
print("------")
print(type(d))
print(d.keys())
print(d['resultCount'])
print("----------------")
print(d['results'][0]['kind'])
# print(a_string['resultCount'])
```

```
{
 "resultCount":25,
 "results": [
{"wrapperType":"track", "kind":"podcast", "collectionId":10892}]}
------
<class 'dict'>
dict_keys(['resultCount', 'results'])
25
----------------
podcast
```

```
import json
def pretty(obj):
    return json.dumps(obj, sort_keys=True, indent=2)

d = {'key1': {'c': True, 'a': 90, '5': 50}, 'key2':{'b': 3, 'c':
"yes"}}
```

```
print(d)
print('--------')
print(pretty(d))
```

```
{'key1': {'c': True, 'a': 90, '5': 50}, 'key2': {'b': 3, 'c': 'yes'}}
--------
{
  "key1": {
    "5": 50,
    "a": 90,
    "c": true
  },
  "key2": {
    "b": 3,
    "c": "yes"
  }
}
```

```
#Say we had a JSON string in the following format. How would you
convert it so that it is a python list?
entertainment = """[{"Library Data": {"count": 3500, "rows": 10,
"locations": 3}}, {"Movie Theater Data": {"count": 8, "rows": 25,
"locations": 2}}]"""
json.loads(entertainment)
```

```
[{'Library Data': {'count': 3500, 'rows': 10, 'locations': 3}},
 {'Movie Theater Data': {'count': 8, 'rows': 25, 'locations': 2}}]
```

```
#Because we can only write strings into a file, if we wanted to
convert a dictionary d into a json-formatted string so that we could
store it in a file, what would we use?

#json.dumps(d)
```

Nested iterations

```
nested1 = [['a', 'b', 'c'],['d', 'e'],['f', 'g', 'h']]
for x in nested1:
    print("level1: ")
    for y in x:
        print("     level2: " + y)
```

```
level1:
     level2: a
     level2: b
     level2: c
level1:
     level2: d
     level2: e
level1:
```

```
    level2: f
    level2: g
    level2: h
```

```
#2. Below, we have provided a list of lists that contain information
about people. Write code to create a new list that contains every
person's last name, and save that list as last_names.
info = [['Tina', 'Turner', 1939, 'singer'], ['Matt', 'Damon', 1970,
'actor'], ['Kristen', 'Wiig', 1973, 'comedian'], ['Michael', 'Phelps',
1985, 'swimmer'], ['Barack', 'Obama', 1961, 'president']]
last_names=[]
for name in info:
  last_names.append(name[1])
print(last_names)
```

```
['Turner', 'Damon', 'Wiig', 'Phelps', 'Obama']
```

```
#Below, we have provided a list of lists named L. Use nested iteration
to save every string containing "b" into a new list named b_strings.
L = [['apples', 'bananas', 'oranges', 'blueberries', 'lemons'],
['carrots', 'peas', 'cucumbers', 'green beans'], ['root beer',
'smoothies', 'cranberry juice']]
b_strings=[]
for lists in L:
    for word in lists:
      if "b" in word:
        b_strings.append(word)
print(b_strings)
```

```
['bananas', 'blueberries', 'cucumbers', 'green beans', 'root beer',
'cranberry juice']
```

Structuring Nested Data

```
# Create a list named nested1 that contains both individual elements
(integers) and sublists
nested1 = [1, 2, ['a', 'b', 'c'], ['d', 'e'], ['f', 'g', 'h']]

# Iterate through each element x in the list nested1
for x in nested1:
    print("level1:")  # This prints "level1:" before checking or
printing the value of x

    # Check if the current element x is a list
    if type(x) is list:
        # If x is a list, iterate through each element y in that
sublist
        for y in x:
            # Print each element y using string formatting
```

```
            print("        level2: {}".format(y))
            # .format(y) replaces the curly braces {} in the string
with the value of y

    else:
        # If x is not a list (i.e., it's an integer like 1 or 2),
print x directly
        print(x)

level1:
1
level1:
2
level1:
        level2: a
        level2: b
        level2: c
level1:
        level2: d
        level2: e
level1:
        level2: f
        level2: g
        level2: h
```

Shallow Copies

```
# Step 1: Define a nested list called `original`
# It contains two sublists: one with dogs/puppies and another with
cats/kittens
original = [['dogs', 'puppies'], ['cats', 'kittens']]

# Step 2: Create a shallow copy of `original` using slicing
# This creates a new outer list but the inner lists are the SAME
references (not copied)
copied_version = original[:]

# Step 3: Print the copied version to see what's inside it
# Output: [['dogs', 'puppies'], ['cats', 'kittens']]
print(copied_version)

# Step 4: Check whether `copied_version` and `original` are the same
object
# This will return False because they are two different outer lists
print(copied_version is original)

# Step 5: Check whether the contents of `copied_version` and
`original` are equal
# This will return True because both lists have the same structure and
contents
```

```
print(copied_version == original)

# Step 6: Modify the original list by appending a new item to the
first sublist
# Since the inner list is shared (shallow copy), this change will
affect both `original` and `copied_version`
original[0].append(["canines"])

# Step 7: Print the modified `original` list
# Output: [['dogs', 'puppies', ['canines']], ['cats', 'kittens']]
print(original)

# Step 8: Print the copied version to observe if it changed too
# Because it's a shallow copy, you'll see the same mutation in the
inner list
print("-------- Now look at the copied version --------")
# Output: [['dogs', 'puppies', ['canines']], ['cats', 'kittens']]
print(copied_version)
```

```
[['dogs', 'puppies'], ['cats', 'kittens']]
False
True
[['dogs', 'puppies', ['canines']], ['cats', 'kittens']]
-------- Now look at the copied version --------
[['dogs', 'puppies', ['canines']], ['cats', 'kittens']]
```

deep copy

```
# Step 1: Define a nested list called 'original'
# It has two inner lists: one with 'dogs' and 'puppies', another with
'cats' and 'kittens'
original = [['dogs', 'puppies'], ['cats', 'kittens']]

# Step 2: Initialize an empty list to store a deep copied version of
'original'
copied_outer_list = []

# Step 3: Start looping through each inner list in the original list
for inner_list in original:
    # Step 4: Create a new list to copy the contents of the current
inner list
    copied_inner_list = []

    # Step 5: Loop through each item in the inner list
    for item in inner_list:
        # Step 6: Add the item to the copied inner list (creates a new
independent list)
        copied_inner_list.append(item)

    # Step 7: After the inner list is copied, add it to the outer
```

```
copied list
    copied_outer_list.append(copied_inner_list)

# Step 8: Print the deep copied version to verify its structure before
modifying the original
print(copied_outer_list)

# Step 9: Now modify the original list: append a new element
['canines'] to the first inner list
original[0].append(['canines'])

# Step 10: Print the modified original to observe the change
print(original)

# Step 11: Print a separator to indicate we're now checking the copied
version
print("-------- Now look at the copied version --------")

# Step 12: Print the deep copied version again
# It will remain unchanged because it holds **independent copies** of
the inner lists
print(copied_outer_list)
```

```
[['dogs', 'puppies'], ['cats', 'kittens']]
[['dogs', 'puppies', ['canines']], ['cats', 'kittens']]
-------- Now look at the copied version --------
[['dogs', 'puppies'], ['cats', 'kittens']]
```

```
# Step 1: Import the built-in copy module which provides deepcopy
functionality
import copy

# Step 2: Define a nested list named 'original'
# It contains:
# - A list: ['canines', ['dogs', 'puppies']]
# - A string: 'felines'
# - A list: ['cats', 'kittens']
original = [['canines', ['dogs', 'puppies']], 'felines', ['cats',
'kittens']]

# Step 3: Create a shallow copy of the original
# This only copies the outer list; inner lists or nested structures
are still shared (referenced)
shallow_copy_version = original[:]

# Step 4: Create a deep copy of the original
# This recursively copies all elements including inner lists and
nested structures,
# so that the new version is fully independent of the original
deeply_copied_version = copy.deepcopy(original)
```

```python
# Step 5: Modify the original list by appending a new string to the
outer list
original.append("Hi there")

# Step 6: Modify an inner list (first element is a list, and its
second element is another list)
# We're appending a new element ['marsupials'] to that deeply nested
list
original[0].append(["marsupials"])

# Step 7: Print the original list to see the modifications
print("-------- Original ------------")
print(original)

# Step 8: Print the deep copied version
# This should NOT include the new changes made to the original list
print("-------- deep copy -----------")
print(deeply_copied_version)

# Step 9: Print the shallow copy
# The outer structure remains the same, but shared nested references
reflect changes made in the original
print("-------- shallow copy -----------")
print(shallow_copy_version)

#look at pythontutor
#essentially original has all the additions. But shallow copy only
referencing parts like [['canines', ['dogs', 'puppies'],
['marsupials']], 'felines', ['cats', 'kittens']]
#so when we add "Hi there" is isnt detecting it. On the otherhand
deepcopy is comepletely detached and is a separate copy without
reference.
```

```
-------- Original ------------
[['canines', ['dogs', 'puppies'], ['marsupials']], 'felines', ['cats',
'kittens'], 'Hi there']
-------- deep copy -----------
[['canines', ['dogs', 'puppies']], 'felines', ['cats', 'kittens']]
-------- shallow copy -----------
[['canines', ['dogs', 'puppies'], ['marsupials']], 'felines', ['cats',
'kittens']]
```

```python
def doubleStuff(a_list):
  new_list=[]
  for value in a_list:
    new_elem=2*value
    new_list.append(new_elem)
  return new_list

things=[2,5,9]
```

```
print(things)
things=doubleStuff(things)
print(things)
```

```
[2, 5, 9]
[4, 10, 18]
```

Using map function

```
def triple(value):
  return 3*value                # Return 3 times the given value

def tripleStuff(a_list):        # Define a function that takes a list
and returns a new list of tripled values
  new_seq=map(triple, a_list)   # Use the map function to apply the
'triple' function to every element in 'a_list'
                                # 'map' returns a map object (an
iterable), which is converted into a list
  return list(new_seq)          # Convert the map object to a list and
return it

things=[2,5,9]
things3=tripleStuff(things)     # Call the function 'tripleStuff' with
'things' as input and store the result in 'things3'
print(things3)                  # Print the resulting list: [6, 15,
27]
```

```
[6, 15, 27]
```

```
celsius=[0,10,20,30]

def celsius_to_fahrenheit(temp):
  return (temp * 9/5) + 32

Fahrenheit_temps= map(celsius_to_fahrenheit, celsius)
#for temp in Fahrenheit_temps:
  #print(temp)
print(list(Fahrenheit_temps))
```

```
[32.0, 50.0, 68.0, 86.0]
```

```
celsius=[0,10,20,30]
Fahrenheit_temps= map(lambda celsius_to_fahrenheit:
(celsius_to_fahrenheit * 9/5) + 32, celsius)
print(list(Fahrenheit_temps))
```

```
[32.0, 50.0, 68.0, 86.0]
```

```
abbrevs = ["usa", "esp", "chn", "jpn", "mex", "can", "rus", "rsa",
"jam"]
```

```
abbrevs_upper=map(lambda upping: (upping.upper()),abbrevs)
print(list(abbrevs_upper))
```

```
['USA', 'ESP', 'CHN', 'JPN', 'MEX', 'CAN', 'RUS', 'RSA', 'JAM']
```

```
abbrevs = ["usa", "esp", "chn", "jpn", "mex", "can", "rus", "rsa",
"jam"]
def f(st):
  return st.upper()

abbrevs_upper2=list(map(f,abbrevs))
print(abbrevs_upper2)
```

```
['USA', 'ESP', 'CHN', 'JPN', 'MEX', 'CAN', 'RUS', 'RSA', 'JAM']
```

Filter function

```
# Define a function that keeps only even numbers from a list
def keep_evens(nums):
    # Use the filter function with a lambda to select only even
numbers
    # lambda num: num % 2 == 0 checks if a number is divisible by 2
(i.e., even)
    new_seq = filter(lambda num: num % 2 == 0, nums)
    return list(new_seq) # Convert the filtered result (a filter
object) to a list and return it

# Call the function with a sample list and print the result
print(keep_evens([3, 4, 6, 7, 0, 1]))  # Output: [4, 6, 0]
```

```
[4, 6, 0]
```

```
lst = ["witch", "halloween", "pumpkin", "cat", "candy", "wagon",
"moon"]
lst2=filter(lambda IsThere0:"o" in IsThere0,lst)
print(list(lst2))
```

```
['halloween', 'wagon', 'moon']
```

List Comprehension

```
things=[2,5,9]
yourlist=[value*2 for value in things]
print(yourlist)
```

```
[4, 10, 18]
```

```
#The full expression for list comprehension can be shown like this:
#[<transformer_expression> for <varname> in <sequence> if
<filtration_expression>]
```

```python
# Define a function called keep_evens that takes a list of numbers as input
def keep_evens(nums):
    # Use list comprehension to build a new list of only even numbers from 'nums'
    # Syntax: [expression for item in iterable if condition]
    # In our case:
    #   expression: num        → what we want to keep in the new list
    #   for num in nums        → loop through each element in 'nums'
    #   if num % 2 == 0        → keep it only if the number is even (divisible by 2)
    new_list = [num for num in nums if num % 2 == 0]

    # Return the new list containing only even numbers
    return new_list

# Call the function with a sample list and print the result
print(keep_evens([3, 4, 6, 7, 0, 1]))  # Output: [4, 6, 0]
```

```
[4, 6, 0]
```

```python
def keep_evens(nums):
  new_list=filter(lambda num: num%2==0,nums)
  return list(new_list)
print(keep_evens([3, 4, 6, 7, 0, 1]))
```

```
[4, 6, 0]
```

```python
# Dictionary named 'tester' contains a key 'info' that maps to a list of dictionaries (like student records)
tester = {
    'info': [
        {'name': "Lauren", 'class standing': 'Junior', 'major': 'Information Science'},
        {'name': "Ayo", 'class standing': 'Junior', 'major': 'Computer Science'},
        {'name': "Kathryn", 'class standing': 'Senior', 'major': 'Linguistics'},
        {'name': "Nick", 'class standing': 'Junior', 'major': 'Information Science'},
        {'name': "Gladys", 'class standing': 'Senior', 'major': 'History'},
        {'name': "Adam", 'class standing': 'Junior', 'major': 'Information Science'}
    ]
}

# Extract the list of dictionaries (student info) from the 'tester' dictionary
inner_list = tester["info"]
```

```
# Use list comprehension to extract just the 'name' values from each
dictionary in the list
# This loops through each dictionary 'd' in inner_list and grabs
d['name']
compri = [d['name'] for d in inner_list]

# Output: ['Lauren', 'Ayo', 'Kathryn', 'Nick', 'Gladys', 'Adam']
print(compri)
```

```
['Lauren', 'Ayo', 'Kathryn', 'Nick', 'Gladys', 'Adam']
```

```
# Import the json module to format output for better readability
import json

# Define a nested dictionary with one key: 'info', which maps to a
list of student records (dictionaries)
tester = {
    'info': [
        {'name': "Lauren", 'class standing': 'Junior', 'major':
'Information Science'},
        {'name': "Ayo", 'class standing': 'Junior', 'major': 'Computer
Science'},
        {'name': "Kathryn", 'class standing': 'Senior', 'major':
'Linguistics'},
        {'name': "Nick", 'class standing': 'Junior', 'major':
'Information Science'},
        {'name': "Gladys", 'class standing': 'Senior', 'major':
'History'},
        {'name': "Adam", 'class standing': 'Junior', 'major':
'Information Science'}
    ]
}

# Extract the list of dictionaries (student data) from the 'tester'
dictionary
inner_list = tester["info"]

# Print the structure of inner_list nicely formatted (for visual
inspection)
print(json.dumps(inner_list, indent=2))  # This is for checking what's
inside

# Use list comprehension to create a new list containing only the
'name' values
# Loop through each dictionary d in inner_list and extract d['name']
compri = [d['name'] for d in inner_list]
print("_____")
print("")
print("")
# Print the final list of names
```

```
print(compri)  # Output: ['Lauren', 'Ayo', 'Kathryn', 'Nick',
'Gladys', 'Adam']
```

```
[
  {
    "name": "Lauren",
    "class standing": "Junior",
    "major": "Information Science"
  },
  {
    "name": "Ayo",
    "class standing": "Junior",
    "major": "Computer Science"
  },
  {
    "name": "Kathryn",
    "class standing": "Senior",
    "major": "Linguistics"
  },
  {
    "name": "Nick",
    "class standing": "Junior",
    "major": "Information Science"
  },
  {
    "name": "Gladys",
    "class standing": "Senior",
    "major": "History"
  },
  {
    "name": "Adam",
    "class standing": "Junior",
    "major": "Information Science"
  }
]
```

```
['Lauren', 'Ayo', 'Kathryn', 'Nick', 'Gladys', 'Adam']
```

```
# Version 1: Using list comprehension
def longlengths(strings):
    # This returns a list of lengths of strings where the string
length is at least 4
    # For each string s in the list, if len(s) >= 4, include len(s) in
the new list
    return [len(s) for s in strings if len(s) >= 4]

# Version 2: Manual method using a loop
def longlengths(strings):
```

```
    accum = []  # Initialize an empty list to store lengths
    for s in strings:
        if len(s) >= 4:
            # If string is long enough, append its length to the list
            accum.append(len(s))
    return accum  # Return the list of lengths

# Version 3: Using filter and map
def longlengths(strings):
    # Use filter to keep only strings with length >= 4
    # lambda s: len(s) >= 4 is an anonymous function used for
filtering
    filtered_strings = filter(lambda s: len(s) >= 4, strings)

    # Use map to apply the len function to each filtered string
    # map returns an iterable of lengths, so wrap it in list() if you
want a list
    return list(map(len, filtered_strings))

# Test the function with a list of strings
# Only 'ghij' (length 4) and 'klmno' (length 5) meet the condition
print(longlengths(['a', 'bc', 'def', 'ghij', 'klmno']))  # Output: [4,
5]
```

```
[4, 5]
```

Zip

```
L1 = [3, 4, 5]
L2 = [1, 2, 3]
L3 = []

# Loop through indices of L1 (and L2 since they're the same length)
for i in range(len(L1)):
    # Add corresponding elements and append the result to L3
    L3.append(L1[i] + L2[i])

# Print the result list
print(L3)
```

```
[4, 6, 8]
```

```
# Two lists of numbers
L1 = [3, 4, 5]
L2 = [1, 2, 3]

# An empty list to store the results
L3 = []

# zip(L1, L2) pairs up elements from both lists: (3,1), (4,2), (5,3)
```

```
# list(zip(...)) converts the zipped object into a list of tuples
L4 = list(zip(L1, L2))  # L4 = [(3, 1), (4, 2), (5, 3)]
print("L4: ", L4)
# Loop through each pair (x1 from L1 and x2 from L2)
for (x1, x2) in L4:
    # Add the paired elements and append the sum to L3
    L3.append(x1 + x2)

# Print the final list of summed values
print(L3)  # Output: [4, 6, 8]
```

```
L4:  [(3, 1), (4, 2), (5, 3)]
[4, 6, 8]
```

```
L1 = [3, 4, 5]
L2 = [1, 2, 3]
L3= [x1+x2 for (x1,x2) in list(zip(L1,L2))]
print(L3)
```

```
[4, 6, 8]
```

```
# Define a function that gives feedback for a guessed word in a
Wordle-like game
def give_feedback(guess, correct_word):
    # Start with an empty list to hold the feedback for each letter
    feedback = []

    # Loop through each index in the guessed word
    for i in range(len(guess)):
        guessed_let = guess[i]         # Get the letter at position i
from the guess
        correct_let = correct_word[i]  # Get the correct letter at the
same position

        # If the guessed letter is exactly correct (right letter in
the right position)
        if guessed_let == correct_let:
            feedback.append("Y")  # Add "Y" to feedback to indicate a
perfect match

        # If the guessed letter is in the correct word but in the
wrong position
        elif guessed_let in correct_word:
            feedback.append("y")  # Add "y" to feedback for right
letter, wrong position

        # If the guessed letter is not in the correct word at all
        else:
            feedback.append("-")  # Add "-" to feedback for wrong
letter
```

```
    # Return the full feedback list
    return feedback
assert list(give_feedback("guess", "guess")) == ['Y', 'Y', 'Y', 'Y',
'Y']
assert list(give_feedback("abcde", "fghij")) == ['-', '-', '-', '-',
'-']
assert list(give_feedback("edcba", "edcba")) == ['Y', 'Y', 'Y', 'Y',
'Y']
assert list(give_feedback("hello", "hails")) == ['Y', '-', '-', '-',
'-']
assert list(give_feedback("tests", "testy")) == ['Y', 'Y', 'Y', 'Y',
'-']
assert list(give_feedback("testy", "tests")) == ['Y', 'Y', 'Y', '-',
'y']
```

```
---------------------------------------------------------------------
-----
AssertionError                            Traceback (most recent call
last)
<ipython-input-40-ee7a5027c2e1> in <cell line: 0>()
     26 assert list(give_feedback("abcde", "fghij")) == ['-', '-',
'-', '-', '-']
     27 assert list(give_feedback("edcba", "edcba")) == ['Y', 'Y',
'Y', 'Y', 'Y']
---> 28 assert list(give_feedback("hello", "hails")) == ['Y', '-',
'-', '-', '-']
     29 assert list(give_feedback("tests", "testy")) == ['Y', 'Y',
'Y', 'Y', '-']
     30 assert list(give_feedback("testy", "tests")) == ['Y', 'Y',
'Y', '-', 'y']

AssertionError:
```

```
# Using zip, it's a little easier to understand
def give_feedback(guess, correct_word):
    # Create an empty list to store feedback for each letter
    feedback = []

    # Use zip() to pair each letter from guess and correct_word
    # For example: zip("hello", "hills") → ('h','h'), ('e','i'),
('l','l'), ...
    for guessed_let, correct_let in zip(guess, correct_word):

        # If the guessed letter is exactly the same as the correct
letter at the same position
        if guessed_let == correct_let:
            feedback.append("Y")  # Add 'Y' to indicate a correct
letter in the correct position

        # If the guessed letter exists somewhere in the correct word
```

```
(but not at this position)
        elif guessed_let in correct_word:
            feedback.append("y")  # Add 'y' to indicate a correct
letter in the wrong position

        # If the guessed letter is not in the correct word at all
        else:
            feedback.append("-")  # Add '-' to indicate the letter is
not in the word

    # Return the complete feedback list
    return feedback
```

Fetching a page (api)

```
import requests  # Import the requests library to make HTTP requests
import json      # Import the json library to handle JSON data

# Make a GET request to the Datamuse API to get words that rhyme with
"funny"
page = requests.get("https://api.datamuse.com/words?rel_rhy=funny")

print(type(page))  # Show the type of the response object (should be
<class 'requests.models.Response'>)
print("------")
print("------")
print("------")
# Print the first 150 characters of the raw response text to get an
idea of the structure
print(page.text[:150])
print("------")
print("------")
print("------")
# Print the URL that was actually fetched (in case it got redirected
or changed)
print(page.url)

print("------")
print("------")
print("------")
# Convert the JSON response text into a native Python object (usually
a list of dictionaries)
x = page.json()

print(type(x))  # Show the type of the converted object (should be a
list)
print("------")
print("------")
print("------")
print("---first item in the list---")
```

```
print(x[0])  # Print the first item from the list of results
print("------")
print("------")
print("------")

print("---the whole list, pretty printed---")
# Pretty-print the full list using json.dumps with indentation
print(json.dumps(x, indent=2))
```

```
<class 'requests.models.Response'>
------
------
------
[{"word":"money","score":4415,"numSyllables":2},
{"word":"honey","score":1206,"numSyllables":2},
{"word":"sunny","score":717,"numSyllables":2},{"word":"
------
------
------
https://api.datamuse.com/words?rel_rhy=funny
------
------
------
<class 'list'>
------
------
------
---first item in the list---
{'word': 'money', 'score': 4415, 'numSyllables': 2}
------
------
------
---the whole list, pretty printed---
[
  {
    "word": "money",
    "score": 4415,
    "numSyllables": 2
  },
  {
    "word": "honey",
    "score": 1206,
    "numSyllables": 2
  },
  {
    "word": "sunny",
    "score": 717,
    "numSyllables": 2
  },
  {
```

```
    "word": "bunny",
    "score": 702,
    "numSyllables": 2
  },
  {
    "word": "blini",
    "score": 613,
    "numSyllables": 2
  },
  {
    "word": "gunny",
    "score": 449,
    "numSyllables": 2
  },
  {
    "word": "tunny",
    "score": 301,
    "numSyllables": 2
  },
  {
    "word": "sonny",
    "score": 286,
    "numSyllables": 2
  },
  {
    "word": "dunny",
    "score": 245,
    "numSyllables": 2
  },
  {
    "word": "runny",
    "score": 225,
    "numSyllables": 2
  },
  {
    "word": "thunny",
    "score": 222,
    "numSyllables": 2
  },
  {
    "word": "aknee",
    "score": 179,
    "numSyllables": 2
  },
  {
    "word": "squinny",
    "score": 170,
    "numSyllables": 2
  },
```

```
  {
    "word": "fiat money",
    "score": 160,
    "numSyllables": 4
  },
  {
    "word": "gunnie",
    "score": 156,
    "numSyllables": 2
  },
  {
    "word": "blood money",
    "score": 152,
    "numSyllables": 3
  },
  {
    "word": "squiny",
    "score": 151,
    "numSyllables": 2
  },
  {
    "word": "tunney",
    "score": 119,
    "numSyllables": 2
  },
  {
    "word": "spinny",
    "score": 117,
    "numSyllables": 2
  },
  {
    "word": "pin money",
    "score": 107,
    "numSyllables": 3
  },
  {
    "word": "easy money",
    "score": 66,
    "numSyllables": 4
  },
  {
    "word": "smart money",
    "score": 66,
    "numSyllables": 3
  },
  {
    "word": "earnest money",
    "score": 62,
    "numSyllables": 4
```

```
  },
  {
    "word": "easter bunny",
    "score": 56,
    "numSyllables": 4
  },
  {
    "word": "paper money",
    "score": 54,
    "numSyllables": 4
  },
  {
    "word": "pocket money",
    "score": 47,
    "numSyllables": 4
  },
  {
    "word": "folding money",
    "score": 46,
    "numSyllables": 4
  },
  {
    "word": "conscience money",
    "score": 41,
    "numSyllables": 4
  },
  {
    "word": "hush money",
    "score": 40,
    "numSyllables": 3
  },
  {
    "word": "prize money",
    "score": 37,
    "numSyllables": 3
  },
  {
    "word": "amount of money",
    "score": 33,
    "numSyllables": 5
  },
  {
    "word": "for love or money",
    "score": 32,
    "numSyllables": 5
  },
  {
    "word": "tight money",
    "score": 32,
```

```
    "numSyllables": 3
  },
  {
    "word": "ship money",
    "score": 30,
    "numSyllables": 3
  },
  {
    "word": "metal money",
    "score": 27,
    "numSyllables": 4
  },
  {
    "word": "sum of money",
    "score": 23,
    "numSyllables": 4
  },
  {
    "word": "entrance money",
    "score": 22,
    "numSyllables": 4
  },
  {
    "word": "cheap money",
    "score": 21,
    "numSyllables": 3
  },
  {
    "word": "spending money",
    "score": 21,
    "numSyllables": 4
  },
  {
    "word": "token money",
    "score": 21,
    "numSyllables": 4
  },
  {
    "word": "waste of money",
    "score": 19,
    "numSyllables": 4
  },
  {
    "word": "ransom money",
    "score": 18,
    "numSyllables": 4
  },
  {
    "word": "hearth money",
```

```
    "score": 14,
    "numSyllables": 3
  },
  {
    "word": "munni",
    "score": 14,
    "numSyllables": 2
  },
  {
    "word": "bunnie",
    "score": 2,
    "numSyllables": 2
  },
  {
    "word": "euromoney",
    "score": 2,
    "numSyllables": 4
  },
  {
    "word": "smartmoney",
    "score": 2,
    "numSyllables": 3
  },
  {
    "word": "anyone he",
    "numSyllables": 4
  },
  {
    "word": "begun he",
    "numSyllables": 3
  },
  {
    "word": "bunney",
    "numSyllables": 2
  },
  {
    "word": "ca ne",
    "numSyllables": 2
  },
  {
    "word": "done he",
    "numSyllables": 2
  },
  {
    "word": "donne e",
    "numSyllables": 2
  },
  {
    "word": "everyone he",
```

```
    "numSyllables": 4
  },
  {
    "word": "fun he",
    "numSyllables": 2
  },
  {
    "word": "grandson he",
    "numSyllables": 3
  },
  {
    "word": "gun he",
    "numSyllables": 2
  },
  {
    "word": "handgun he",
    "numSyllables": 3
  },
  {
    "word": "kun hee",
    "numSyllables": 2
  },
  {
    "word": "le ne",
    "numSyllables": 2
  },
  {
    "word": "lunney",
    "numSyllables": 2
  },
  {
    "word": "lunny",
    "numSyllables": 2
  },
  {
    "word": "none e",
    "numSyllables": 2
  },
  {
    "word": "none he",
    "numSyllables": 2
  },
  {
    "word": "nun he",
    "numSyllables": 2
  },
  {
    "word": "one he",
    "numSyllables": 2
```

```
  },
  {
    "word": "one knee",
    "numSyllables": 2
  },
  {
    "word": "pun he",
    "numSyllables": 2
  },
  {
    "word": "run e",
    "numSyllables": 2
  },
  {
    "word": "run he",
    "numSyllables": 2
  },
  {
    "word": "shotgun he",
    "numSyllables": 3
  },
  {
    "word": "someone e",
    "numSyllables": 3
  },
  {
    "word": "someone he",
    "numSyllables": 3
  },
  {
    "word": "son e",
    "numSyllables": 2
  },
  {
    "word": "son he",
    "numSyllables": 2
  },
  {
    "word": "sun e",
    "numSyllables": 2
  },
  {
    "word": "sun he",
    "numSyllables": 2
  },
  {
    "word": "ton he",
    "numSyllables": 2
  },
```

```
  {
    "word": "ton ne",
    "numSyllables": 2
  },
  {
    "word": "un e",
    "numSyllables": 2
  },
  {
    "word": "un he",
    "numSyllables": 2
  },
  {
    "word": "un ne",
    "numSyllables": 2
  },
  {
    "word": "un ni",
    "numSyllables": 2
  },
  {
    "word": "won he",
    "numSyllables": 2
  }
]
```

```python
# Import statements for necessary Python modules
import requests  # This lets us send HTTP requests to websites/APIs

# Define a function to get rhyming words
def get_rhymes(word):
    # This is the base URL of the API we're calling
    baseurl = "https://api.datamuse.com/words"

    # Create an empty dictionary to store URL parameters
    params_diction = {}

    # Add a key 'rel_rhy' with the input word as the value
    # This tells the API to find rhymes for the word
    params_diction["rel_rhy"] = word

    # Limit the number of results to 3
    params_diction["max"] = "3"

    # Send a GET request to the API with the base URL and parameters
    resp = requests.get(baseurl, params=params_diction)

    # Convert the JSON response into a Python object (list of
dictionaries)
    word_ds = resp.json()
```

```
    # Use list comprehension to extract only the 'word' from each
dictionary
    return [d['word'] for d in word_ds]

# Call the function and print rhyming words for 'funny'
print(get_rhymes("funny"))

# Call the function and print rhyming words for 'dash'
print(get_rhymes("dash"))
```

```
['money', 'honey', 'sunny']
['cache', 'flash', 'ash']
```

```
import requests
import json

PERMANENT_CACHE_FNAME = "./assets/permanent_cache.txt"
TEMP_CACHE_FNAME = "this_page_cache.txt"

def _write_to_file(cache, fname):
    with open(fname, 'w') as outfile:
        outfile.write(json.dumps(cache, indent=2))

def _read_from_file(fname):
    try:
        with open(fname, 'r') as infile:
            res = infile.read()
            return json.loads(res)
    except:
        return {}

def add_to_cache(cache_file, cache_key, cache_value):
    temp_cache = _read_from_file(cache_file)
    temp_cache[cache_key] = cache_value
    _write_to_file(temp_cache, cache_file)

def clear_cache(cache_file=TEMP_CACHE_FNAME):
    _write_to_file({}, cache_file)

def requestURL(baseurl, params = {}):
    # This function accepts a URL path and a params diction as inputs.
    # It calls requests.get() with those inputs,
    # and returns the full URL of the data you want to get.
    req = requests.Request(method = 'GET', url = baseurl, params =
params)
    prepped = req.prepare()
    return prepped.url
```

```
def make_cache_key(baseurl, params_d, private_keys=["api_key",
"apikey"]):
    """Makes a long string representing the query.
    Alphabetize the keys from the params dictionary so we get the same
order each time.
    Omit keys with private info."""
    alphabetized_keys = sorted(params_d.keys())
    res = []
    for k in alphabetized_keys:
        if k not in private_keys:
            val = params_d[k]
            if type(val) == type([]):
                val = ','.join([str(item) for item in val])
                print(val)
            res.append("{}-{}".format(k, val))
    return baseurl + "_".join(res)

def perm_cache():
    return _read_from_file(PERMANENT_CACHE_FNAME)

def get(baseurl, params={}, private_keys_to_ignore=["api_key",
"apikey"], permanent_cache_file=PERMANENT_CACHE_FNAME,
temp_cache_file=TEMP_CACHE_FNAME, headers=None):
    full_url = requestURL(baseurl, params)
    cache_key = make_cache_key(baseurl, params,
private_keys_to_ignore)
    # Load the permanent and page-specific caches from files
    permanent_cache = _read_from_file(permanent_cache_file)
    temp_cache = _read_from_file(temp_cache_file)
    if cache_key in temp_cache:
        print("found in page-specific cache")
        # make a Response object containing text from the change, and
the full_url that would have been fetched
        resp = requests.Response()
        resp._content = bytes(temp_cache[cache_key], 'utf-8')
        resp.url = full_url
        return resp
    elif cache_key in permanent_cache:
        print("found in permanent_cache")
        # make a Response object containing text from the change, and
the full_url that would have been fetched
        resp = requests.Response()
        resp._content = bytes(permanent_cache[cache_key], 'utf-8')
        resp.url = full_url
        return resp
    else:
        print("new; adding to cache")
        # actually request it
        resp = requests.get(baseurl, params, headers=headers)
```

```
        # save it
        add_to_cache(temp_cache_file, cache_key, resp.text)
        return resp
#---------
#------------------------------
#-------------------------------------------------------------
import requests
import json
import os

# This is the file we will use to store the cache
CACHE_FILENAME = "cache_file.json"

# Load cache if it exists
if os.path.exists(CACHE_FILENAME):
    with open(CACHE_FILENAME, 'r') as file:
        cache = json.load(file)
else:
    cache = {}

def get_with_caching(url):
    """Returns cached data if available, otherwise makes a request and
saves it"""
    if url in cache:
        print("Getting from cache...")
        return cache[url]
    else:
        print("Fetching from the web...")
        response = requests.get(url)
        cache[url] = response.text
        with open(CACHE_FILENAME, 'w') as file:
            json.dump(cache, file)
        return response.text

# Example usage
url = "https://api.datamuse.com/words?rel_rhy=happy"
data = get_with_caching(url)
print(data[:100])

#_____________________________________________________________________
_____________________________________________
#_____________________________________________________________________
_____________________________________________
# Import a special version of the requests library that includes
caching functionality
import requests_with_caching

# First request: the response is not yet in the cache, so it is
fetched from the web
res = requests_with_caching.get(
```

```
    "https://api.datamuse.com/words?rel_rhy=happy",  # API request for
words that rhyme with "happy"
    permanent_cache_file="datamuse_cache.txt"        # Specify a file
to permanently store cached responses
)

# Print the first 100 characters of the response text to see part of
the result
print(res.text[:100])

# Second request: same URL as before, so this result will be fetched
from the temporary cache
res = requests_with_caching.get(
    "https://api.datamuse.com/words?rel_rhy=happy",
    permanent_cache_file="datamuse_cache.txt"
)

# Third request: different word ("funny"), might be in the permanent
cache if fetched before
res = requests_with_caching.get(
    "https://api.datamuse.com/words?rel_rhy=funny",
    permanent_cache_file="datamuse_cache.txt"
)

Getting from cache...
[{"word":"nappy","score":703,"numSyllables":2},
{"word":"snappy","score":698,"numSyllables":2},{"word

---------------------------------------------------------------------
-----
ModuleNotFoundError                       Traceback (most recent call
last)
<ipython-input-52-5e80f7d51c75> in <cell line: 0>()
    116
#_____________________________________________________________________
_____________________________________
    117 # Import a special version of the requests library that
includes caching functionality
--> 118 import requests_with_caching
    119
    120 # First request: the response is not yet in the cache, so it
is fetched from the web

ModuleNotFoundError: No module named 'requests_with_caching'

---------------------------------------------------------------------
-----
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.
```

```
To view examples of installing some common dependencies, click the
"Open Examples" button below.
--------------------------------------------------------------------------
-----


import requests
import json
import os

# File where the cached responses will be stored
CACHE_FILENAME = "cache_file.json"

# Load the cache from file if it exists, otherwise use an empty
dictionary
if os.path.exists(CACHE_FILENAME):
    with open(CACHE_FILENAME, 'r') as f:
        cache = json.load(f)
else:
    cache = {}

def get_with_caching(url):
    """
    Returns the cached response for a given URL if it exists.
    Otherwise, makes a request, caches it, and returns the result.
    """
    if url in cache:
        print("Getting from cache...")
        return cache[url]
    else:
        print("Fetching from the web...")
        response = requests.get(url)
        data = response.text
        cache[url] = data
        # Save updated cache to file
        with open(CACHE_FILENAME, 'w') as f:
            json.dump(cache, f)
        return data

# □ Example Usage
url1 = "https://api.datamuse.com/words?rel_rhy=happy"
url2 = "https://api.datamuse.com/words?rel_rhy=funny"

# First request – fetched from the web
data1 = get_with_caching(url1)
print(data1[:100])  # Print only the first 100 characters

# Second request – same URL, should be fetched from cache
data1_again = get_with_caching(url1)
print(data1_again[:100])
```

```
# Third request – different word, likely new
data2 = get_with_caching(url2)
print(data2[:100])
```

```
Getting from cache...
[{"word":"nappy","score":703,"numSyllables":2},
{"word":"snappy","score":698,"numSyllables":2},{"word
Getting from cache...
[{"word":"nappy","score":703,"numSyllables":2},
{"word":"snappy","score":698,"numSyllables":2},{"word
Fetching from the web...
[{"word":"money","score":4415,"numSyllables":2},
{"word":"honey","score":1206,"numSyllables":2},{"wor
```

```
# import statements
import requests_with_caching
import json
# import webbrowser

# apply for a flickr authentication key at
http://www.flickr.com/services/apps/create/apply/?
# paste the key (not the secret) as the value of the variable
flickr_key
flickr_key = 'yourkeyhere'

def get_flickr_data(tags_string):
    baseurl = "https://api.flickr.com/services/rest/"
    params_diction = {}
    params_diction["api_key"] = flickr_key # from the above global
variable
    params_diction["tags"] = tags_string # must be a comma separated
string to work correctly
    params_diction["tag_mode"] = "all"
    params_diction["method"] = "flickr.photos.search"
    params_diction["per_page"] = 5
    params_diction["media"] = "photos"
    params_diction["format"] = "json"
    params_diction["nojsoncallback"] = 1
    flickr_resp = requests_with_caching.get(baseurl, params =
params_diction, permanent_cache_file="flickr_cache.txt")
    # Useful for debugging: print the url! Uncomment the below line to
do so.
    print(flickr_resp.url) # Paste the result into the browser to
check it out...
    return flickr_resp.json()

result_river_mts = get_flickr_data("river,mountains")

# Some code to open up a few photos that are tagged with the mountains
```

```
and river tags...

photos = result_river_mts['photos']['photo']
for photo in photos:
    owner = photo['owner']
    photo_id = photo['id']
    url = 'https://www.flickr.com/photos/{}/{}'.format(owner,
photo_id)
    print(url)
    # webbrowser.open(url)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call
last)
<ipython-input-48-fd849ff51cc8> in <cell line: 0>()
      1 # import statements
----> 2 import requests_with_caching
      3 import json
      4 # import webbrowser
      5

ModuleNotFoundError: No module named 'requests_with_caching'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

```
import requests
import json

API_KEY = "PASTE_YOUR_FLICKR_API_KEY_HERE"

base_url = "https://api.flickr.com/services/rest/"
params = {
    "method": "flickr.photos.search",
    "format": "json",
    "nojsoncallback": 1,
    "per_page": 5,
    "tags": "mountains,river",
    "tag_mode": "all",
    "media": "photos",
    "api_key": API_KEY
```

```
}

response = requests.get(base_url, params=params)

if response.status_code == 200:
    print("✅ Successfully retrieved data from Flickr API.\n")

    data = response.json()
    print("📦 JSON response:")
    print(json.dumps(data, indent=2))  # This shows what's inside

    # Now try to extract photos if the 'photos' key exists
    if 'photos' in data:
        photos = data['photos']['photo']
        print("\n🖼 Displaying photo links:\n")
        for photo in photos:
            owner = photo['owner']
            photo_id = photo['id']
            photo_url =
f"https://www.flickr.com/photos/{owner}/{photo_id}"
            print(photo_url)
    else:
        print("\n⚠️ 'photos' key not found. Likely an API error:")
        if 'message' in data:
            print("❌ Flickr Error:", data['message'])
        else:
            print(data)
else:
    print("❌ Failed to retrieve data. HTTP Status:",
response.status_code)
```

```
✅ Successfully retrieved data from Flickr API.

📦 JSON response:
{
  "stat": "fail",
  "code": 100,
  "message": "Invalid API Key (Key has invalid format)"
}

⚠️ 'photos' key not found. Likely an API error:
❌ Flickr Error: Invalid API Key (Key has invalid format)
```