Mehedi Azad
Data Analyst | Data Science | Machine Learning | Statistics |
AI | Power BI | Tableau | Pandas | Numpy | SQL | Python |

1. Apply Functions to Elements in a List

any: Check if Any Element of an Iterable is True

If you want to check if any element of an iterable is True, use any.

In the code below, I use any to find if any element in the text is in uppercase.

In [2]:
```
text = 'abcdeF'
any(x for x in text if x.isupper())
```
Out[2]:
```
True
```

all: Check if All Elements of an Interable Are Strings

If you want to check if all elements of an iterable are strings, use all and isinstance.

In [7]:
```
# for string
l = ['a', 'b', 3, 2]
all(isinstance(item, str) for item in l)
```
Out[7]:
```
False
```

In [8]:
```
# for integer
i = [0, 2, 1, 3, 2]
all(isinstance(item, int) for item in i)
```
Out[8]:
```
True
```

filter: Get the Elements of an Iterable that a Function Returns True

If you want to get the elements of an iterable that a function returns true, use filter.

In the code below, I use the filter method to get items that are fruits.

In [21]:
```
def get_fruit(val:str):
    fruits = ['apple', 'orange', 'banana']
    if val in fruits:
        return True
    else:
        return False

items = ['chair', 'apple', 'water', 'table', 'banana']
fruits = filter(get_fruit, items)
list(fruits)
```
Out[21]:
```
['apple', 'banana']
```

In [28]:
```
# we can use this function for any types
def demo_func(val):
    items = ['apple', 'orange', 'banana', 1, 2]
    if val in items:
        return True
    else:
        return False

items2 = ['chair', 'apple', 'water', 'table', 'banana',1,2,3,4,5]
items = filter(demo_func, items2)
list(items)
```

Out[28]:
```
['apple', 'banana', 1, 2]
```

map method: Apply a Function to Each Item of an Iterable

If you want to apply the given function to each item of a given iterable, use map.

In [48]:
```
nums = [1,2,3,4]
list(map(str, nums))  # map() must have at least two arguments.
```

Out[48]:
```
['1', '2', '3', '4']
```

In [38]:
```
def multiply_by_two(num: float):
    return num*2
list(map(multiply_by_two, nums))
```

Out[38]:
```
[2, 4, 6, 8]
```

2. Get Elements

random.choice: Get a Randomly Selected Element from a Python List

Besides getting a random number, you can also get a random element from a Python list using random.

In the code below, 'do excercise' was picked randomly from a list of options.

In [41]:
```
import random

to_do_tonight = ['stay at home', 'attend a party', 'do excercise']
random.choice(to_do_tonight)
```

Out[41]:
```
'do excercise'
```

random.sample: Get Multiple Random Elements from a Python List

If you want to get n random elements from a list, use random.sample.

```
In [50]: import random

         random.seed(1)    # The seed() method is used to initialize the random number ger
         nums = [1,2,3,4,5,6,7,8,9,10]
         random.sample(nums, 3)
```

```
Out[50]: [3, 2, 5]
```

3. Good Practices

Stop using = operator to create a copy of a Python list.

Use copy method instead

When you create a copy of a Python list using the = operator,

a change in the new list will lead to the change in the old list. It is because both lists point to the same object.

```
In [54]: l1= [1,2,3,4,5]
         l2 = l1
         l2.append(6)
```

```
In [57]: print(l2)
         print(l1)
```

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

```
In [60]: # Instead of using = operator, use copy() method.
         # Now your old list will not change when you change your new list.
```

```
In [62]: l1 = [1,2,3,4]
         l2 = l1.copy()
         l2.append(5)
```

```
In [63]: print(l2)
         print(l1)
```

```
[1, 2, 3, 4, 5]
[1, 2, 3, 4]
```

Enumerate: Get Counter and Value While Looping

Are you using for i in range(len(array)) to access both the index and the value of the array?

If so, use enumerate instead. It produces the same result but it is much cleaner.

In [64]:
```python
arr = ['a', 'b', 'c', 'd', 'e']

# instead of this
for i in range(len(arr)):
    print (i, arr[i])
```

```
0 a
1 b
2 c
3 d
4 e
```

In [65]:
```python
# use this
for i, val in enumerate(arr):
    print(i, val)
```

```
0 a
1 b
2 c
3 d
4 e
```

Difference between list append and list extend

If you want to add a list to another list, use the append() method.

To add elements of a list to another list, use the extend() method.

In [66]:
```python
# Add a list to a list
a = [1, 2, 3, 4]
a.append([5, 6])
a
```

Out[66]:
```
[1, 2, 3, 4, [5, 6]]
```

In [68]:
```python
# Add elements of a list to another list
a = [1, 2, 3 ,4]
a.extend([5, 6])
a
```

Out[68]:
```
[1, 2, 3, 4, 5, 6]
```

4. Interaction Between 2 Lists

set.intersection: Find the Intersection Between 2 Sets

If you want to get the common elements between 2 lists, convert lists

to sets then use set.intersection to find the intersection between 2 sets.

In [72]:
```python
requirement1 = ['pandas', 'numpy', 'matplotlib']
requirement2 = ['numpy', 'seaborn', 'plotly', 'pandas']

intersection = set.intersection(set(requirement1), set(requirement2))
list(intersection)
```

Out[72]:
```
['pandas', 'numpy']
```

Set Difference: Find the Difference Between 2 Sets

If you want to find the difference between 2 lists, turn

those lists into sets then apply the difference() method to the sets.

In [1]:
```python
a = [1, 2, 3, 4]
b = [1, 3, 4, 5, 6]
```

In [4]:
```python
# Find elements in 'list a' but not in 'list b'
diff1 = set(a).difference(set(b))
print(list(diff1))
```

```
[2]
```

In [5]:
```python
# Find elements in 'list b' but not in 'list a'
diff2 = set(b).difference(set(a))
print(list(diff2))
```

```
[5, 6]
```

5. Join Iterables

join method: Turn an Iterable into a Python String

If you want to turn an iterable into a string, use join().

In the code below, I join elements in the list fruits using “, “.

In [12]:
```python
fruits = ['apples', 'orange', 'bananas']
fruits_str = ', '.join(fruits)
print(f'Today, i need to get some {fruits_str} in the grocery store')
```

```
Today, i need to get some apples, orange, bananas in the grocery store
```

Zip: Associate Elements from Two Iterators based on the Order

If you want to associate elements from two iterators based on the order, combine list and zip.

```
In [17]: nums = [1, 2, 3, 4, 5]
         string = 'abcde'
         list(zip(nums, string))

Out[17]: [(1, 'a'), (2, 'b'), (3, 'c'), (4, 'd'), (5, 'e')]
```

Zip Function: Create Pairs of Elements from Two Lists in Python

If you want to create pairs of elements from two lists, use zip. zip() function takes

iterables and aggregates them in a tuple.

You can also unzip the list of tuples by using zip(*list_of_tuples).

```
In [33]: nums = [1, 2, 3, 4]
         chars = ['a', 'b', 'c', 'd']
         comb = list(zip(nums, chars))
         comb

Out[33]: [(1, 'a'), (2, 'b'), (3, 'c'), (4, 'd')]
```

```
In [34]: # unzip the list of tuples by using zip(*list_of_tuples).
         list(zip(*comb))

Out[34]: [(1, 2, 3, 4), ('a', 'b', 'c', 'd')]
```

6. Unpack Iterables

How to Unpack Iterables in Python

To assign items of a Python iterables (such as list, tuple, string) to different variables,

you can unpack the iterable like below.

```
In [54]: nasted_arr = [[1, 2, 3], ['a', 'b'], 4]
         # num_arr, char_arr, num = nasted_arr
         nasted_arr = num_arr, char_arr, num
```

```
In [55]: print(num_arr)
         print(char_arr)
         print(num)

         [1, 2, 3]
         ['a', 'b']
         4
```

Extended Iterable Unpacking: Ignore Multiple Values

when Unpacking a Python Iterable

If you want to ignore multiple values when unpacking a Python iterable, add * to _ as shown below.

This is called “Extended Iterable Unpacking” and >>>>>>>is available in Python 3.x.

In [62]:
```
a, *_, b = [1, 2, 3, 4]
print(a)
print(*_)
print(b)
```

```
1
2 3
4
```

In [ ]:
```
```

Mehedi Azdad
Data Analyst | Data Science | Machine Learning | Statistics | AI | Power BI | Tableau |
Pandas | Numpy | SQL | Python |

https://www.linkedin.com/in/mehediazad/