1)

In [1]:
```
# [] is an empty list. An empty list is used to store values when a function or code is run.

a = []

b = [1,2,3,4,5,6,7,8,9]
for i in b:
    if i % 2 == 0:
        a.append(i)
print(a)

'''
In the above example, the empty list a stores the even values,
when the following snippet of code is run.
'''
```

```
[2, 4, 6, 8]
```

2)

In [3]:
```
spam = [2,4,6,8,10]
spam.insert(2,'hello')
print(spam)
```

```
[2, 4, 'hello', 6, 8, 10]
```

3)

In [4]:
```
spam = ['a','b','c','d']
spam[int(int('3' * 2) / 11)]

# gives the index 3 value
```

Out[4]:
```
'd'
```

4)

In [6]:
```
spam = ['a','b','c','d']
print(spam[-1])
```

```
d
```

5)

In [7]:
```
spam = ['a','b','c','d']
print(spam[:2])
```

```
['a', 'b']
```

6)

In [8]:
```
bacon = [3.14,'cat',11,'cat',True]
bacon.index('cat')
```

Out[8]:
```
1
```

7)

In [12]:
```
bacon = [3.14,'cat',11,'cat',True]
bacon.append(99)
bacon

# .append() adds values to the list
```

Out[12]:
```
[3.14, 'cat', 11, 'cat', True, 99]
```

8)

In [14]:
```
bacon = [3.14,'cat',11,'cat',True]
bacon.remove('cat')
bacon

'''
.remove() function removes values form a list.
In the above list, cat is repeated tewo time,
.remove() deleted the first occurence of cat in the list.
'''
```

Out[14]:
```
[3.14, 11, 'cat', True]
```

9)

In [16]:
```
# List Concatenation

list1 = [1,2,3,4]
list2 = [5,6,7,8]

list1.extend(list2)  # concats list2 to list1
print9list1
```

Out[16]:
```
[1, 2, 3, 4, 5, 6, 7, 8]
```

In [17]:
```
list1 = [1,2,3,4]
list2 = [5,6,7,8]

list3 = list1 + list2 #list3 contains concated list1 and list2
print(list3)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

In [19]:
```
# list replication

list4 = list1.copy()
list4
```

Out[19]:
```
[1, 2, 3, 4]
```

10)

In [27]:
```
# append()

list1 = [1,2,3,4]
list1.append(5)
print('Append:', list1) # append() adds elements to the end of the list4
list1.insert(1,1.5)
print('Insert:', list1) # with insert() we can insert element anywhere in the list
```

```
Append: [1, 2, 3, 4, 5]
Insert: [1, 1.5, 2, 3, 4, 5]
```

11)

In [39]:
```
# Methods to remove elements from the list

list4 = [1,2,3,4,5,6,7,8,9]

list4.remove(2) # remove() will remove 2 from the list

list4.pop(3) #pop() will remove 3rd index element from the list

del list4[5] #del will remove 5th index element from the list

list4
```

Out[39]:
```
[1, 3, 4, 6, 7, 9]
```

12)

In [45]:
```
# List
my_list = [1, 2, 3, 4, 5]

# String
my_string = "Hello, World!"

# Accessing elements using indexing
print(my_list[2])         # Output: 3
print(my_string[7])       # Output: W
```

```
3
W
```

In [46]:
```
# Slicing
print(my_list[1:4])       # Output: [2, 3, 4]
print(my_string[0:5])     # Output: Hello
```

```
[2, 3, 4]
Hello
```

In [47]:
```
# Iterating through elements
for item in my_list:
    print(item)

for char in my_string:
    print(char)
```

```
1
2
3
4
5
H
e
l
l
o
,

W
o
r
l
d
!
```

13)

In [ ]:
```
'''
Mutability:
Lists: Lists are mutable, which means you can add, remove, or modify elements after the list is created.
You can change the size and content of a list during the program's execution.
Tuples: Tuples, on the other hand, are immutable. Once a tuple is created, you cannot change its elements or size.
If you need a collection that should not be modified, tuples are a better choice.

Syntax:

Lists: Lists are defined using square brackets [].
Tuples: Tuples are defined using parentheses ().
'''
```

14)

In [64]:
```
tup = (42,)
type(tup)
```

Out[64]:
```
tuple
```

15)

In [65]:
```
my_list = [1, 2, 3, 4, 5]
my_tuple = tuple(my_list)
print(my_tuple)  # list to tuple
```

```
(1, 2, 3, 4, 5)
```

In [68]:
```
my_tuple2 = (1,2,3,4,5,6)
my_list2 = list(my_tuple2)
print(my_list2) # tuple to list
```

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

16)

In [ ]:
```
'''
Variables that "contain" list values in Python are not lists themselves.
Instead, they contain references to the list objects in memory.
In Python, when you assign a list (or any other mutable object) to a variable,
the variable stores a reference to the memory location where the actual list data is stored.
This is true for all mutable objects, including lists, dictionaries, sets, etc.
'''
```

17)

In [77]:
```
# copy.copy() (Shallow Copy):
# Changes to the nested objects in the shallow copy will affect the original and vice versa.

import copy
original_list = [1, [2, 3], 4]
shallow_copy = copy.copy(original_list)

shallow_copy[1][0] = 99
print('original:',original_list)
print('copied:',shallow_copy)
```

```
original: [1, [99, 3], 4]
copied: [1, [99, 3], 4]
```

In [78]:
```
# copy.deepcopy() (Deep Copy):
# Changes to the nested objects in the deep copy will not affect the original and vice versa.
# Deep copy ensures that the copied object and all its nested objects are completely independent from the original

import copy

original_list = [1, [2, 3], 4]
deep_copy = copy.deepcopy(original_list)

deep_copy[1][0] = 99
print('original:',original_list)
print('copied:',deep_copy)
```

```
original: [1, [2, 3], 4]
copied: [1, [99, 3], 4]
```

In [ ]:
```

```