In [1]:
```
mylist = [1,2,3]
```

In [2]:
```
mylist.append(4)
```

In [3]:
```
mylist
```

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

In [4]:
```
mylist.pop()
```

Out[4]:
```
4
```

In [5]:
```
mylist
```

Out[5]:
```
[1, 2, 3]
```

In [6]:
```
help(mylist.insert)
```

```
Help on built-in function insert:

insert(index, object, /) method of builtins.list instance
    Insert object before index.
```

Functions: Allow us to create blocks of code that can be easily eecuted many times.without needing to constantly reqrite the entire block of code

def name_of_function(name):

```
'''
Docstring explains function.
'''
print("hello" + name)
```

> name_of_function(name) Hello Jose

In [10]:
```
def say_hello():
    print('Hello')
    print('are')
    print('you')
```

In [11]:
```
say_hello() # say_hello is the function name
```

```
Hello
```

```
are
you
```

In [14]:
```
def say_hello(name):
    print(f'Hello {name}')
```

In [15]:
```
say_hello('Jose')
```

```
Hello Jose
```

In [18]:
```
def add_num(num1,num2):
    return num1+num2
```

In [20]:
```
result= add_num(10,20)  # YOU CAN SAVE THE WHOLE FUNCTION AS A VARIABLE OR ASSIGN IT TO
```

In [21]:
```
result
```

Out[21]:
```
30
```

In [22]:
```
def print_result(a,b):
    print(a+b)
```

In [23]:
```
def return_result(a,b):
    return a+b
```

In [25]:
```
result = print_result(10,20)
```

```
30
```

In [26]:
```
result
```

In [27]:
```
type(result)
```

Out[27]:
```
NoneType
```

In [28]:
```
return_result(10,20)
```

Out[28]:
```
30
```

In [31]:
```
result = return_result(10,20)
```

In [32]:
```
result
```

Out[32]:
```
30
```

In [33]:
```
def myfunc(a,b):
    print(a+b)
    return a + b
```

In [34]:
```
result = myfunc(10,20)
```

```
30
```

In [35]:
```
result
```

Out[35]:
```
30
```

In [36]:
```
def sum_numbers(num1,num2):
    return num1 + num2
```

In [37]:
```
sum_numbers(10,20)
```

Out[37]:
```
30
```

In [38]:
```
sum_numbers('10','20') # BE CAREFUL BECAUSE IT IS NOT DEFINED THEY JUST GET ADDED TOGET
```

Out[38]:
```
'1020'
```

45. FUNCTIONS WITH LOGIC

In [39]:
```
2 % 2
```

Out[39]:
```
0
```

In [40]:
```
3 % 2  # MOD OPERATOR JUST RETURNS THE REMAINDER OF A DIVISION
```

Out[40]:
```
1
```

In [42]:
```
20%2   #20 is DIVISIBLE BY 2 so REMAINDER IS 0
```

Out[42]:
```
0
```

In [43]:
```
20 % 2 == 0
```

Out[43]:
```
True
```

In [44]:
```
def even_check(number):
    result = number % 2 == 0
```

```
    return result
```

In [45]:
```
even_check(20)
```

Out[45]:
```
True
```

In [46]:
```
even_check(21)
```

Out[46]:
```
False
```

In [47]:
```
# RETURN TRUE IF ANY NUMBER IS EVEN INSIDE OF A LIST
```

In [64]:
```
def check_even_list(num_list):
    #return all even numbers in a list

    #placeholder variables
    even_numbers = []

    for number in num_list:
        if number % 2 == 0:
            even_numbers.append(number)
        else:
            pass
    return even_numbers
```

In [65]:
```
check_even_list([1,2,3,4,5])
```

Out[65]:
```
[2, 4]
```

In [66]:
```
check_even_list([1,3,5])
```

Out[66]:
```
[]
```

In [67]:
```
check_even_list([2,1,1,1])
```

Out[67]:
```
[2]
```

46. TUPLE UNPACKING WITH PYTHON FUNCTIONS

In [ ]:
```
```