CLOSURES

Simply put, calling a nested function outside of a scope.

see: function nesting; global, local and nonlocal scopes; first class functions.

Since we know that.  
1. Inner\_functions can access variables from outer\_functions even after execution is completed(returned ).  
2. Functions can be returned(as functions/as variables) and assigned to variables  
3. Calling nested functions outside of the enclosing scope returns an error

Method to create a Closure:

Objective: Calling the Nested\_function/Inner\_function/enclosed\_function, outside of its scope by assigning the Outer\_function/enclosing\_function to a variable(bounds) and calling the variable as a function with (parameters). Provided the Outer\_function returns the Inner\_function as a variable w/o (parameters).

1. Define Parent: Define a function(parent) with a parameter with placeholder argument,
2. Define Child: Define a nested function(child) within it and use the argument from the parent\_function
3. Return Child: After defining the child function, Return it without paranthesis inside the parent.
4. Bound Parent: Assign the parent function with the parameters to a variable
5. Closure: call the variable with parameters anywhere you want.. you got yourself a Closure Object.

def outer(*name*):

    # this is the enclosing function

    print(*name*)

    def inner():

        # this is the enclosed function

        # the inner function accessing the outer function's variable 'name'

        print('hello',*name*)

    return inner

#call the enclosing function by assigning it to a variable

#the enclosing function returns (function\_object: inner) to (variable : hello). So that hello=inner

hello=outer('Closure')

#calling the enclosed/nested function by calling the variable with paranthesis, hello() -> inner()

hello()

#But directly calling the nested function, returns an error.

inner()

You can even delete the entire function using (keyword: del) and still call the function using closures,

hello()

print ( outer)

del outer

hello()

print ( outer)

hello()

We access the contents of the closure, gives a tuple  
this dunder is an attribute that applies to any function object

#dunder to access it's contents

print(hello.\_\_closure\_\_)

print(hello.\_\_closure\_\_[0].cell\_contents)

Another simplified Closure example::.,

# An simple closure

def parent(*child*):

        def *child*():

                print('Closure has been called')

        return *child*

my\_closure=parent('a\_string\_passer\_that\_takes\_the\_palce\_of\_Child')

print(parent)#checking if parent function exists

del parent

# print(parent) The closure exists even after the parent has been deleted

my\_closure()