METHODS

Methods perform specific actions on an object & can also take arguments.

They are on the form:

object.method(arg1, arg2 ---)

For e.g.

l=[1,2,3,4,5]

You can see all possible methods by pressing the tab key.
Press shift+Tab to get more help about the method.

~~For~~

In: help(l.count)

Help on built-in function count:

count(...) method of builtins.list instance
    L.count(value) → integer -- return number of occurences of value

FUNCTIONS

Formally, a func^n is a useful device that groups together a set of statements

so they can be run more than once.
They also let us specify parameters that serve inputs to the funcns.
It helps us to reuse code.

def keyword

Syntax:

~~def me~~ snake case
def name_of_function(arg1, arg2):
    '''
    This is where the function's Document String (docstring) goes.
    When you call help() on your funcn it would be printed out.
    '''
    #Do stuff here
    #Return desired result

→ We begin with def keyword then a space followed by name of function.
→ You shouldn't call a function with same name as built in funcn in Python.
→ In the pair of parentheses write the nos. of arguments seperated by comma.
→ You must indent the code ~~to begin~~ properly.
→ In the docstring you write the basic description of the function.

→ Using Jupyter Notebooks, you'll be able to read these docstrings by pressing Shift + Tab after func" name.

→ Docstrings are not necessary for simple func"s but it's a good practice to use them so that others understand your code.

E.g.

```
qn: def say_hello():
        print('hello')
```

Calling a func" with ()

```
qn: say_hello()
o/p: 'hello'
```

If you forget the parentheses

```
qn: say_hello
o/p: <function main.say_hello>
```

It simply displays the fact that say_hello is a func"

With parameters

```
qn: def greeting(name):
        print(f'Hello {name}')
```

Date
Page

In: greeting('Jose')

o/p: Hello Jose

Returns → It allows the func^n to return a result which can be stored as a variable.

Eg.

~~In: deff~~

In: def add_num(n1, n2):
    return n1 + n2

In: add_num(4,5)

o/p: 9

In: res = add_num(4, 5)
print(res)

~~o/p:~~

o/p: 9

In: add_num('one', 'two')

o/p: 'onetwo'

"Difference betⁿ return & print:

The return keyword allows you to

actually save the result of the o/p func^n as a variable & but the print func^n does not.

Q For eg.

qn: def print_result(a,b):
    print(a+b)

qn: def return_result(a,b):
    return a+b

qn: my_res = print_result(20,20)

o/p: 40

qn: type(my_res)

o/p: NoneType

qn: my_res = return_result(20,20)

qn: my_res

o/p: 40

Functions can also return tuples

qn: work_hours = [('Abby', 100), ('Billy', 400), ('Cassie', 800)]

The employee of the ~~math~~ month func" will return both the name & no. of hours worked for the top performer (Judged by no. of hours worked)

qn: def employee_check (work_hours):

    c_m = 0

    e_m = ''

    ~~for~~ for employee, hours in work_hours:
        if hours > ~~current~~ c_m:
            c_m = hours
            e_m = employee
        else:
            pass

    return (e_m, c_m)

qn: employee_check (work_hours)

o/p: ('Cassie', 800)

qn: name, hours = employee_check (work_hours)

qn: name

o/p: Cassie

qn: hours

o/p: 800