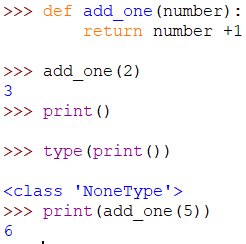
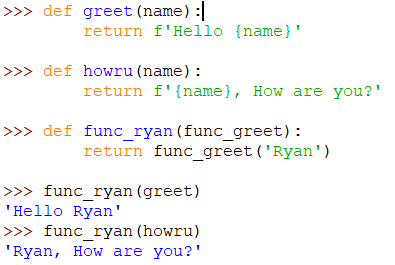
Decorators wrap a function, modifying its behaviour.

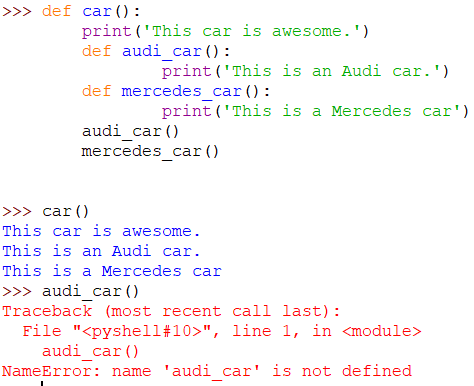
1. A *function* returns a value based on the given arguments. Decorators are functions those turn the arguments into values.
2. In Python, functions are *first-class objects*. This means that functions can be passed around and used as arguments, just like any other object (string, int, float, list, and so on).

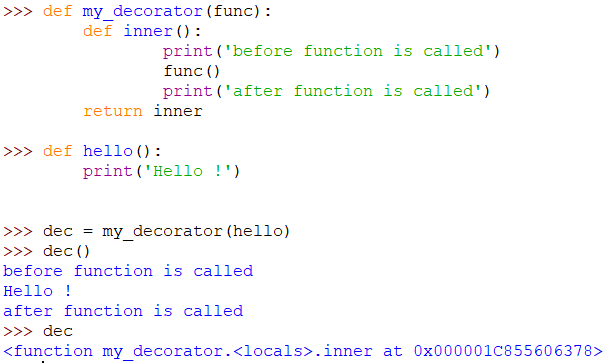


 Here is another example.

In the example**, greet()** & **howru()** are regular functions seeking a string argument to return values. But **func\_ryan()**  isseeking a function to be passed as an argument.

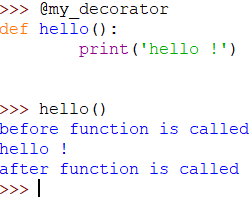
1. Functions inside other functions are called *inner functions.* In a function, inner functions don’t follow orders. Like other functions these are executed. The inner functions are locally scoped to outer functions. Hence can’t be called separately. An outer function can return inner function.



Above concepts lay foundation for implementing decorators in Python code block.

1. In the above example we have implemented the concepts of first class objects, inner function. The *dec* represents the so-called decoration here. In effect, the name *dec* now points to the inner() inner function. Remember that you return *inner* as a function when you call *my\_decorator(hello).* However, *inner*() has a reference to the original *hello*() as func, and calls that function between the two calls to *print*().

**The pie syntax(@):**

1. Instead of writing *dec = my\_decorator(hello)* you can write @my\_decorators.

**Reusing decorators:**

1. Let’s create a python file my\_decorator.py with below code blocks.

def reuse\_decor(func):

def wrapper\_reuse (\*args, \*\*kwargs):

func(\*args, \*\*kwargs)

func(\*args, \*\*kwargs)

return wrapper\_reuse

create a greet.py using below code blocks.

from my\_decorator import reuse\_decor

@ reuse\_decor

def greet(name):

print(f"Hello {name}")

create hey.py using below code blocks.

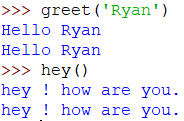
from my\_decorator import wrapper\_reuse

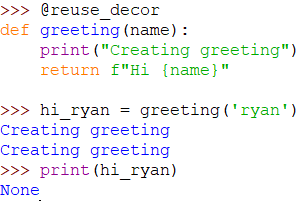
@ reuse\_decor

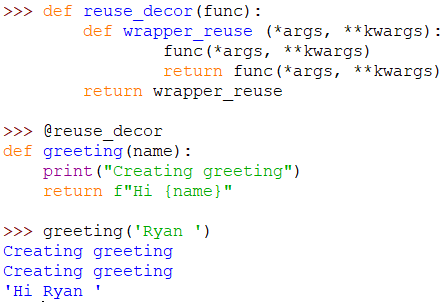
def hey():

print(‘Hey ! how are you.’)

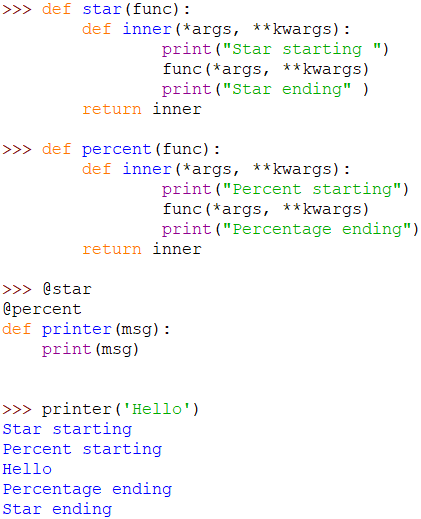
Since the my\_decorator.py uses two inner functions so below will be the output. Here ***functions are decorated with arguments***

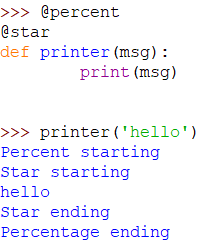


1. If you see below code, the return doesn’t work with decorators.

Then what is the other way out! You just make the inner function return the value of decorated function.

Chaining the decorators



The output order changes with order of decorator tagging.