## Learning site links:

<https://medium.com/edureka/python-decorator-tutorial-bf7b21278564>

<https://www.programiz.com/python-programming/decorator>

<https://www.learnbyexample.org/python-decorators/>

<https://www.studytonight.com/python/python-decorators>

<https://pybit.es/decorators-by-example.html>

<https://www.saltycrane.com/blog/2010/03/simple-python-decorator-examples/>

<https://dbader.org/blog/python-decorators>

<https://www.geeksforgeeks.org/decorators-in-python/>

<https://realpython.com/primer-on-python-decorators/>

<https://godatadriven.com/blog/a-practical-example-of-generators-and-decorators-in-python/>

<https://www.techbeamers.com/python-decorator/>

<https://www.datacamp.com/community/tutorials/decorators-python>

## example code:

*def* make\_pretty(*func*):

*def* inner():

        print("I got decorated")

        func()

    return inner

*def* ordinary():

    print("I am ordinary")

#pretty = make\_pretty(ordinary)

#pretty()

ordinary = make\_pretty(ordinary)

#ordinary()

#@make\_pretty

#def ordinary():

#    print("I am ordinary")

#ordinary()

#def ordinary():

#    print("I am ordinary")

#ordinary = make\_pretty(ordinary)

#ordinary()

*def* fun1(*xx*):

*def* fun2():

        print('i am fun2')

        xx()

    return fun2()

*def* fun3():

    print('i am fun3')

fun3=fun1(fun3)

*def* one(*yy*):

*def* two():

        print('i am two')

        yy()

    return two()

    #def four():

    #    print('i am four')

    #    yy()

    #return four()

@one

*def* three():

    print('i am three')

*def* one(*yy*):

*def* two():

        print('i am two before')

        yy()

        print('i am two after')

    return two()

@one

*def* three():

    print('i am three')

*def* outer():

    print('outer')

*def* inner():

        print("Running inner")

    inner()

outer()

*def* outer():

    #print('outer')

*def* inner(*name*):

        print("Hello", name)

    return inner

sos = outer()

sos("Bob")

*def* decorate\_it(*func*):

*def* wrapper():

        print("Before function call")

        func()

        print("After function call")

    return wrapper

@decorate\_it

*def* hello():

    print("Hello world")

hello()

*def* one(*msg*):

    print(msg)

# calling the function

one("Hello!")

# having a new variable reference it

two = one

# calling the new variable

two("Hello!")

three=two

three('hello')

# a decorator function in genaral formate

*def* myDecor(*func*):

    # inner function like in closures

*def* wrapper():

        print("Modified function")

        func()

    return wrapper

*def* myfunc():

    print('Hello!!')

# Calling myfunc()

myfunc()

# decorating the myfunc function

decorated\_myfunc = myDecor(myfunc)

# calling the decorated version

decorated\_myfunc()

# a decorator function with decorate formate

*def* myDecor(*func*):

    # inner function like in closures

*def* wrapper():

        print("Modified function")

        func()

    return wrapper

@myDecor

*def* myfunc():

    print('Hello!!')

myfunc()

*def* one(*x*):

    print('first run')

    return x

*def* three(*y*):

    print('third run')

    return y

*def* four(*z*):

    print('fourth run')

    return z

@four

@three

@one

*def* two():

    print('second run')

two()

#not work decoretor

#def decoratorFunc(fn):

    #return 10

#@decoratorFunc

#def existingFunc():

    #print("Hello World!")

#existingFunc()

#right way to decoretor

*def* decoratorFunc(*fn*):

*def* callExistingFunc():

        print("%s was called." % fn)

        fn()

    return callExistingFunc

@decoratorFunc

*def* existingFunc():

    print("Hello World!")

existingFunc()

*def* helloworld(*ob*):

    print ("Hello world")

    return ob

*def* myfunc():

    print ("my function")

myfuns=helloworld(myfunc)

myfunc()

#using decorator

*def* helloworld(*ob*):

    print ("Hello world")

    return ob

@helloworld

*def* myfunc():

    print ("my function")

myfunc()

*def* one(*x*):

    print('first run')

    return x

*def* three(*y*):

    print('third run')

    return y

*def* four(*z*):

    print('fourth run')

    return z

@four

@three

@one

*def* two():

    print('second run')

two()

#muliple decoreor example

*def* fun1(*x*):

    print('i am fun1')

*def* one():

        print('i am one in fun1')

        x()

    return one

*def* fun2(*y*):

    print('i am fun2')

*def* two():

        print('i am two in fun2')

        y()

    return two

@fun1

@fun2

*def* cool():

    print('hello all')

cool()

*def* parent(*num*):

*def* first\_child():

        return "Hi, I am Emma"

*def* second\_child():

        return "Call me Liam"

    if num == 1:

        return first\_child

    else:

        return second\_child

first = parent(1)

print(first())

second = parent(3)

print(second())

try:

*def* decorate\_it(*func*):

*def* wrapper():

            print("Before function call")

            func()

            print("After function call")

        return wrapper

    @decorate\_it

*def* hello(*name*):

        print("Hello", name)

    hello("Bob")

#..............................................

except:

*def* decorate\_it(*func*):

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

            print("Before function call")

            func(\*args, \*\*kwargs)

            print("After function call")

        return wrapper

    @decorate\_it

*def* hello(*name*):

        print("Hello", name)

    hello("Bob")

    # Prints Before function call

    # Prints Hello Bob

    # Prints After function call

*def* outer(*x*):

    print('i am outer')

*def* one(*args*, \*\**kwargs*):

        print('outer in one before')

        x(args, \*\*kwargs)

        print('outer in one after')

    return one

@outer

*def* cool(*name*):

    print('i am '+name)

cool('mahmdu')

*def* outer(*x*):

    #print('i am outer')

*def* one(*args*, \*\**kwargs*):

        print('outer in one before')

        x(args, \*\*kwargs)

        print('outer in one after')

    return one

*def* inner(*x*):

    #print('i am inner')

*def* one(*args*, \*\**kwargs*):

        print('inner in one before')

        x(args, \*\*kwargs)

        print('inner in one after')

    return one

@outer

@inner

*def* cool(*name*):

    print('i am '+name)

cool('mahmdu')

*def* star(*func*):

*def* inner(\**args*, \*\**kwargs*):

        print("\*" \* 30)

        func(\*args, \*\*kwargs)

        print("\*" \* 10)

    return inner

*def* percent(*func*):

*def* inner(\**args*, \*\**kwargs*):

        print("%" \* 30)

        func(\*args, \*\*kwargs)

        print("%" \* 10)

    return inner

@star

@percent

*def* printer(*msg*):

    print(msg)

printer("Hello")

from functools import wraps

*def* mydecorator(*f*):

    @wraps(f)

*def* wrapped(\**args*, \*\**kwargs*):

        print ("Before decorated function")

        r = f(\*args, \*\*kwargs)

        print ("After decorated function")

        return r

    return wrapped

@mydecorator

*def* myfunc(*myarg*):

    print ("my function", myarg)

    return "return value"

r = myfunc('asdf')

print (r)

## practical example ,use decorator in django project

<https://levelup.gitconnected.com/how-to-implement-login-logout-and-registration-with-djangos-user-model-59442164db73>

**\*\*\*Make sure to apply migrations to create the tables to store the User information.**

## example 1, initial setup step:

use login only home page

basic setting need pro/urls, urls.py, views.py, login.html,

### pro/url:

from django.contrib import admin

from django.urls import path,include

urlpatterns = [

    path('admin/', admin.site.urls),

    path('',include('app1.urls')),

    path('accounts/',include('django.contrib.auth.urls')), #new

]

### Urls.py

from django.urls import path

from . import views

urlpatterns = [

    path('',views.one,name="home"),

    path('base/',views.two,name="base"),

]

### Views.py

from django.shortcuts import render

from django.contrib.auth.decorators import login\_required #new

@login\_required

def one(request):

    return render(request,'home.html')

### templates/registration/Login.html

<form method="post">

    {% csrf\_token %}

    {{form.as\_p}}

    <input type="submit" value="Login">

    </form>

### Home.html

<h1>home</h1>

## Example 2, update 1 step

Need login before multiple page (home,base,…..)

Setting.py, pro/url,login.html, home.html stile .just add

### Urls.py

from django.urls import path

from . import views

urlpatterns = [

    path('',views.one,name="home"),

    path('base/',views.two,name="base"),  #new

]

### Views.py

from django.shortcuts import render

from django.contrib.auth.decorators import login\_required

@login\_required

def one(request):

    return render(request,'home.html')

@login\_required #new

def two(request):

    return render(request,'base.html')

### Base.html

<h1> I am bas </h1>

## Example 3, update step 2. Add registration/sign\_up step

Ned login to entry home page , if not login ,goto sign\_up page to registrations then login to home page.

### Setting.py

LOGIN\_REDIRECT\_URL = 'home' #new

### Pro/urls

from django.contrib import admin

from django.urls import path,include

urlpatterns = [

    path('admin/', admin.site.urls),

    path('',include('app1.urls')),

    path('accounts/',include('django.contrib.auth.urls')), #new

]

### Urls.py

from django.urls import path

from . import views

urlpatterns = [

    path('',views.one,name="home"),

    path('base/',views.two,name="base"),

    path('accounts/sign\_up/',views.sign\_up,name="sign-up"), #new

]

### Views.py

from django.urls import path

from . import views

urlpatterns = [

    path('',views.one,name="home"),

    path('base/',views.two,name="base"),

    path('accounts/sign\_up/',views.sign\_up,name="sign-up"), #new

]

### registrations/Login.html

<form method="post">

    {% csrf\_token %}

    {{form.as\_p}}

    <input type="submit" value="Login">

    </form>

### registrations /Sign-up.html

### templates/home.html

### templaes/base.html