Higher Order Functions(Daha Yüksek Dereceli Fonksiyonlar)

1- Bir fonksiyon bir veya daha fazla fonksiyonu parametre olarak alabilir.

2- Bir işlev, başka bir işlevin sonucu olarak döndürülebilir.

3- Bir fonksiyon değiştirilebilir

4-Bir değişkene bir fonksiyon atanabilir

5-Bu bölümde şunları ele alacağız:

* Fonksiyonları parametre olarak işleme
* Fonksiyonları başka fonksiyonlardan dönüş değeri olarak döndürme
* Python kapaklarını ve dekoratörlerini kullanma

## Function as a Parameter(Fonksiyon Parametreleri)

def sum\_numbers(nums):  # normal fonksiyon

    return sum(nums)

Yukarıda Pythonda normal bir fonksiyon tanımladığımızı ve parametreleri görebiliriz.

## Function as a Return Value(Dönüş değeri Fonksiyonu)

Farklı fonksiyonlar dönüş değerinde kullanılablir. Bazı örnek fonksiyonları inceleyelim.

def square(x):

    return x \*\* 2       # a karesel fonksiyon

def cube(x):            # a kübik fonksiyon

    return x \*\* 3

def absolute(x):        # an mutlak değer fonksiyon

    if x >= 0:

        return x

    else:

        return -(x)

# Python Closures(Python Kapanışları)

Python, iç içe geçmiş bir işlevin, çevreleyen işlevin dış kapsamına erişmesine izin verir. Bu Kapanış olarak bilinir. Python'da kapanışların nasıl çalıştığına bir göz atalım. Python'da, bir işlevi başka bir kapsülleme işlevinin içine yerleştirip ardından iç işlevi döndürerek kapatmış oluruz.

def add\_eight():

    eight = 8

    def add(num):

        return num + eight

    return add

closure\_result = add\_eight()

print(closure\_result(5))  # 5+8 13

print(closure\_result(10)) # 10+8 18

# Python Decorators(Python Dekoretörleri)

Dekoratör, Python'da, kullanıcının yapısını değiştirmeden mevcut bir nesneye yeni işlevler eklemesine izin veren bir tasarım kalıbıdır. Dekoratörler genellikle süslemek istediğiniz bir fonksiyonun tanımından önce çağrılır.

def greeting():

    return 'I try learning Python'

def uppercase\_decorator(function):

    def wrapper():

        func = function()

        make\_uppercase = func.upper()

        return make\_uppercase

    return wrapper

g = uppercase\_decorator(greeting)

print(g())

#I TRY LEARNING PYTHON

## Applying Multiple Decorators to a Single Function(Tek Bir İşleve Birden Çok Dekoratör Uygulamak)

* İlk Dekoratör

def uppercase\_decorator(function):

    def wrapper():

        func = function()

        make\_uppercase = func.upper()

        return make\_uppercase

    return wrapper

* İkinci Dekoratör

def split\_string\_decorator(function):

    def wrapper():

        func = function()

        splitted\_string = func.split()

        return splitted\_string

    return wrapper

## Accepting Parameters in Decorator Functions

(Dekoratör Fonksiyonlarında Parametreleri Kabul Etme)

Çoğu zaman parametreleri almak için fonksiyonlarımıza ihtiyaç duyarız, bu yüzden parametreleri kabul eden bir dekoratör tanımlamamız gerekebilir.

def decorator\_with\_parameters(function):

    def wrapper\_accepting\_parameters(para1, para2, para3):

        function(para1, para2, para3)

        print("I live in {}".format(para3))

    return wrapper\_accepting\_parameters

# Built-in Higher Order Functions(Yerleşik Yüksek Dereceli Fonksiyonlar)

Bu bölümde ele aldığımız yerleşik yüksek dereceli işlevlerden bazıları map() , filter ve Reduce'dir . Lambda işlevi parametre olarak geçirilebilir ve lambda işlevlerinin en iyi kullanım durumu harita, filtre ve küçültme gibi işlevlerdedir.

## Python - Map Function(Python - Harita İşlevi)

map() işlevi, bir işlevi alan ve parametre olarak yinelenebilen yerleşik bir işlevdir.

Örnek:

numbers\_str = ['1', '2', '3', '4', '5']

numbers\_int = map(int, numbers\_str)

print(list(numbers\_int))    # [1, 2, 3, 4, 5]

Örnek:

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

def square(x):

    return x \*\* 2

numbers\_squared = map(square, numbers)

print(list(numbers\_squared))    # [1, 4, 9, 16, 25]

# Aynı işlemi lambda ile yapabiliriz.

numbers\_squared = map(lambda x : x \*\* 2, numbers)

print(list(numbers\_squared))    # [1, 4, 9, 16, 25]

map ile liste üzerinde değişiklik yaparak tekrar döndürebiliriz.

## Python - Filter Function(Filtre Fonksiyonu)

filter() işlevi, belirtilen yinelenebilir öğenin (liste) her öğesi için boole değeri döndüren belirtilen işlevi çağırır. Filtreleme kriterlerini karşılayan öğeleri filtreler.

Örnek:

numbers = [1, 2, 3, 4, 5,6]

def is\_even(num):

    if num % 2 == 0:

        return True

    return False

even\_numbers = filter(is\_even, numbers)

print(list(even\_numbers))       # [2, 4 , 6]

Örnek:

names = ['Fatma Büşra', 'Beyza', 'Kübra', 'Beste']  # iterable

def is\_name\_long(name):

    if len(name) > 7:

        return True

    return False

long\_names = filter(is\_name\_long, names)

print(list(long\_names))

#['Fatma Büşra']

## Python - Reduce Function(Fonksiyonu Azaltır)

reduce () fonksiyonu functools modülünde tanımlanan ve bu modülü alınacak gerekir. Harita ve filtre gibi, bir işlev ve bir yinelenebilir olmak üzere iki parametre alır. Ancak, başka bir yinelenebilir döndürmez, bunun yerine tek bir değer döndürür.

numbers\_str = ['1', '2', '3', '4', '5']  # iterable

def add\_two\_nums(x, y):

    return int(x) + int(y)

total = reduce(add\_two\_nums, numbers\_str)

print(total) #15