Introduction

Bu derste, kodunuzu fonksiyonlarla nasıl düzenleyeceğinizi öğreneceksiniz. Fonksiyon, belirli bir görevi gerçekleştirmek üzere tasarlanmış bir kod bloğudur. Göreceğiniz gibi, fonksiyonlar hiçbir kodu tekrarlamadan hemen hemen aynı hesaplamayı birden çok kez yapmanıza olanak tanır.

# Intro to functions: a simple example

Basit bir fonksiyon örneğiyle başlayalım. Aşağıdaki add\_three() fonksiyonu herhangi bir sayıyı alır, ona üç ekler ve ardından sonucu döndürür.

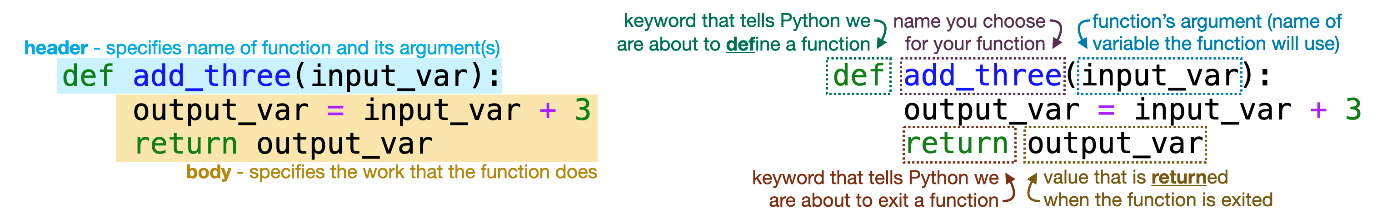
*# Define the function*

def add\_three(input\_var):

output\_var = input\_var + 3

return output\_var

Her fonksiyon iki parçadan oluşur: Başlık ve gövde.



## **Header**

Fonksiyon başlığı, fonksiyonun adını ve argümanlarını tanımlar.

Her fonksiyon başlığı, Python'a bir fonksiyon tanımlamak üzere olduğumuzu bildiren def ile başlar.

Örnekteki fonksiyon adı add\_three'dir.

Örnekteki argüman input\_var'dır. Argüman, fonksiyona girdi olarak kullanılacak değişkenin adıdır. Her zaman fonksiyon adından hemen sonra görünen parantezlerin içine alınır. (Bir fonksiyonun hiç argümanı olmayabileceğini veya birden fazla argümanı olabileceğini unutmayın. Bunun bazı örneklerini derste daha sonra göreceksiniz.)

Her fonksiyon için, fonksiyon argümanlarını çevreleyen parantezlerin ardından iki nokta üst üste : gelmelidir.

## **Body**

Fonksiyon gövdesi, fonksiyonun yaptığı işi belirtir.

Fonksiyon gövdesindeki her kod satırı tam dört boşluk girintili olmalıdır. Bunu boşluk tuşuna dört kez basarak veya klavyenizdeki "Tab" tuşuna bir kez basarak yapabilirsiniz. (Python hakkında daha fazla bilgi edindikçe kodunuzu dört boşluktan fazla girintili yapmanız gerekebilir, ancak bu konuda daha fazla bilgiyi bu kursun ilerleyen bölümlerinde öğreneceksiniz.)

Fonksiyon, girintili tüm satırları yukarıdan aşağıya doğru çalıştırarak işini yapar.

Örnekteki input\_var argümanı input olarak alır.

Fonksiyon, output\_var = input\_var + 3 hesaplamasıyla yeni bir output\_var değişkeni oluşturur.

Sonra, return ifadesi adı verilen son kod satırı, output\_var içindeki değeri fonksiyonun çıktısı olarak döndürür.

Yukarıdaki kod hücresi fonksiyonu yalnızca tanımlar, ancak çalıştırmaz. Fonksiyon gövdesinin ayrıntıları, fonksiyonu gerçekten çalıştırdığımız bir sonraki kod hücresinden sonra daha anlamlı hale gelecektir.

# How to run (or "call") a function

Bir fonksiyonu çalıştırdığımızda, buna fonksiyonu "çağırmak" da denebilir.

Aşağıdaki kod hücresinde, fonksiyonu 10 giriş değeriyle çalıştırıyoruz. Fonksiyonun çıktısına atanacak yeni bir değişken olan new\_number tanımlıyoruz.

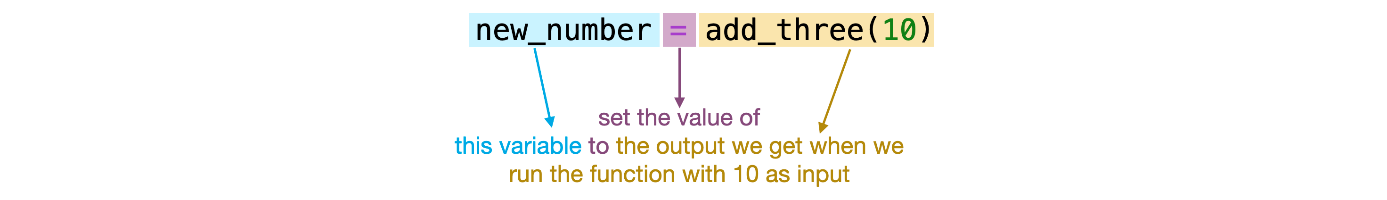
*# Run the function with 10 as input*

new\_number = add\_three(10)

*# Check that the value is 13, as expected*

print(new\_number)

13



Daha detaylı olarak,

add\_three(10), input\_var için 10 değerini girip add\_three() fonksiyonunu çağırdığımızda çıktı olarak aldığımız değerdir. Fonksiyon çalıştığında, gövdesindeki tüm kodu yukarıdan aşağıya doğru çalıştırır:

Önce output\_var = input\_var + 3'ü hesaplar ve bu da output\_var değerini 13 olarak ayarlar.

Kodun son satırı, output\_var değerinin 13 olan değerini döndüren return ifadesidir.

new\_number = add\_three(10) değerini ayarlayarak new\_number değerini 13 olarak ayarlarız.

Not: Bu eğitimde add\_three() fonksiyonuna gelişigüzel atıfta bulunduğumuzda, fonksiyon adından sonra boş parantez kullanırız. Bu, insanların Python kod açıklamalarını genellikle yazma biçimiyle tutarlıdır ve boş parantezler, bir değişkenden veya başka bir Python nesnesinden ziyade bir fonksiyondan bahsettiğimizi açıkça belirtir. Fonksiyonun argümanları olsa bile, bu parantezler her zaman boş olmalıdır.

# Naming functions

Yukarıdaki örnekte, fonksiyonun adı sizin için seçilmiştir. Kendi fonksiyonlarınıza isim verirken, yalnızca küçük harfler kullanmalı ve kelimeleri boşluk yerine alt çizgiyle ayırmalısınız.

Fonksiyonlara isim vermek zamanla doğal gelecektir ve ilk başta rahatsız edici gelmesi normaldir. Öğrenmenin en iyi yolu, bolca Python kodu incelemektir.

# A more complex example

Artık temelleri anladığınıza göre, daha uzun bir hesaplama örneğiyle devam edebiliriz.

Diyelim ki bir arkadaşınızın vergi sonrası haftalık maaşını hesaplamasına yardım ediyorsunuz.

%12 vergi dilimindeler (başka bir deyişle, maaşlarının %12'si vergi olarak alınıyor ve eve sadece %88'ini götürüyorlar) ve

Saatlik olarak, saatte 15 dolar üzerinden ücret alıyorlar.

Aşağıdaki fonksiyon, maaşı çalışılan saat sayısına göre hesaplar. Fonksiyon, ilk örnektekinden daha karmaşıktır çünkü fonksiyonda daha fazla kod satırı ve açıklama vardır. Yukarıdaki örneğe benzer şekilde, fonksiyonun tek bir argümanı vardır (num hours). Fonksiyon gövdesinde şunları yaparız:

num\_hours değerini, yeni bir pay\_pretax değişkeninin değerini belirtmek için kullanırız.

pay\_pretax değerini, yeni bir pay\_aftertax değişkeninin değerini belirtmek için kullanırız.

pay\_aftertax değişkeninin değerini döndürürüz.

def get\_pay(num\_hours):

*# Pre-tax pay, based on receiving $15/hour*

pay\_pretax = num\_hours \* 15

*# After-tax pay, based on being in 12% tax bracket*

pay\_aftertax = pay\_pretax \* (1 - .12)

return pay\_aftertax

Bu fonksiyonu, ilk fonksiyonla aynı şekilde adlandırıyoruz. Bir sonraki kod hücresi, 40 saat çalışmaya göre maaşı hesaplar. (Vergiler düşüldükten sonra 528 dolardır.)

*# Calculate pay based on working 40 hours*

pay\_fulltime = get\_pay(40)

print(pay\_fulltime)

528.0

Çalışılan farklı saat sayısına göre ücreti hızlıca hesaplamak için, fonksiyona farklı bir sayı girmeniz gerekir. Örneğin, arkadaşınızın 32 saat çalıştığını varsayalım. (Sonra 422,40 dolar alır.)

pay\_parttime = get\_pay(32)

print(pay\_parttime)

422.4

Bir fonksiyon yazdığınız için, hesaplamalardaki tüm kodu tekrar yazmak zorunda kalmadan farklı saatler için ücret hesaplayabilirsiniz.

Fonksiyonlar, kodunuzda hatalardan kaçınmanıza yardımcı olabilir ve size çok zaman kazandırır. Genel olarak, kod yazarken mümkün olduğunca az yazmaya çalışmalısınız, çünkü her hesaplama yazdığınızda, yanlışlıkla bir yazım hatası veya hata yapma olasılığınız artar.

# Variable "scope"

Fonksiyon gövdesi içinde tanımlanan değişkenlere fonksiyon dışından erişilemez. Örneğin, bir sonraki kod hücresi hatalıdır, çünkü pay\_aftertax yalnızca fonksiyonun içinde mevcuttur.

print(pay\_aftertax)

---------------------------------------------------------------------------

NameError Traceback (most recent call last)

/tmp/ipykernel\_19/2991332200.py in <module>

----> 1 print(pay\_aftertax)

NameError: name 'pay\_aftertax' is not defined

pay\_pretax veya num\_hours değerlerini yazdırmaya çalışırsanız aynı hatayı alırsınız. Bu nedenle, bir fonksiyondan herhangi bir bilgiye ihtiyacınız varsa, bu bilginin fonksiyonun sonundaki return ifadesinde göründüğünden emin olmanız gerekir.

Bir değişkenin kapsamını, kodun erişilebilir kısmı olarak tanımlarız. Bir fonksiyon içinde tanımlanan değişkenler (pay\_aftertax gibi), yalnızca o fonksiyonun yerel kapsamına sahiptir. Ancak, gördüğünüz gibi, tüm fonksiyonların dışında tanımlanan değişkenler (pay\_parttime gibi) genel bir kapsama sahiptir ve her yerden erişilebilir.

# Functions with multiple arguments

Şimdiye kadar, tek bir argümanla bir fonksiyonu nasıl tanımlayacağınızı öğrendiniz. Birden fazla argümanlı bir fonksiyon tanımlamak için, fonksiyon başlığındaki parantezlerin içine daha fazla argüman eklemeniz ve bunları virgülle ayırmanız yeterlidir.

Bunu, haftalık maaş bordrosunu üç argümana göre hesaplayan aşağıdaki get\_pay\_with\_more\_inputs() fonksiyonuyla yapıyoruz:

num\_hours - bir haftada çalışılan saat sayısı

hourly\_wage - saatlik ücret ($/saat cinsinden)

tax\_bracket - maaşınızdan vergiler için düşülen yüzde

def get\_pay\_with\_more\_inputs(num\_hours, hourly\_wage, tax\_bracket):

*# Pre-tax pay*

pay\_pretax = num\_hours \* hourly\_wage

*# After-tax pay*

pay\_aftertax = pay\_pretax \* (1 - tax\_bracket)

return pay\_aftertax

Ardından, fonksiyonu çağırmak için her girdi için yine virgülle ayrılmış bir değer sağlamanız gerekir.

Aşağıdaki kod hücresinde, 40 saat çalışan, saatte 24 dolar kazanan ve %22 vergi diliminde olan birinin vergi sonrası maaşını hesaplıyoruz.

higher\_pay\_aftertax = get\_pay\_with\_more\_inputs(40, 24, .22)

print(higher\_pay\_aftertax)

748.8000000000001

Aşağıdaki kod hücresi, get\_pay(40)'ı çalıştırdığımızda aldığımız sonucun aynısını verir, çünkü hourly\_wage 15'e, tax\_bracket ise %12'ye ayarlanmıştır ve bu da get\_pay'i tasarlama şeklimizle örtüşmektedir.

same\_pay\_fulltime = get\_pay\_with\_more\_inputs(40, 15, .12)

print(same\_pay\_fulltime)

528.0

Bu yeni get\_pay\_with\_more\_inputs() işlevini nasıl kullanmayı planladığımıza bağlı olarak, daha fazla durumu ele aldığı için orijinal get\_pay() işlevinden daha faydalı olabilir. Yeni işlev, saatlik ücret ve vergi dilimini hatalı bir şekilde varsaymak yerine, kullanıcının doğru değerleri belirlemesine olanak tanır. Ancak, saatlik ücret ve vergi diliminin değişmesi gerekmeyeceğinden eminseniz, yeni işlev gereğinden fazla karmaşıktır. Genel olarak, işlevleri tanımlarken, kullanım durumunuza bağlı olarak ne kadar esnekliğe ihtiyacınız olduğunu göz önünde bulundurmanız gerekir.

# Functions with no arguments

Argümansız ve return ifadesi olmayan fonksiyonlar tanımlamanın mümkün olduğunu unutmayın. Aşağıdaki kod hücresindeki print\_hello() fonksiyonu buna bir örnektir.

*# Define the function with no arguments and with no return*

def print\_hello():

print("Hello, you!")

print("Good morning!")

*# Call the function*

print\_hello()

Hello, you!

Good morning!