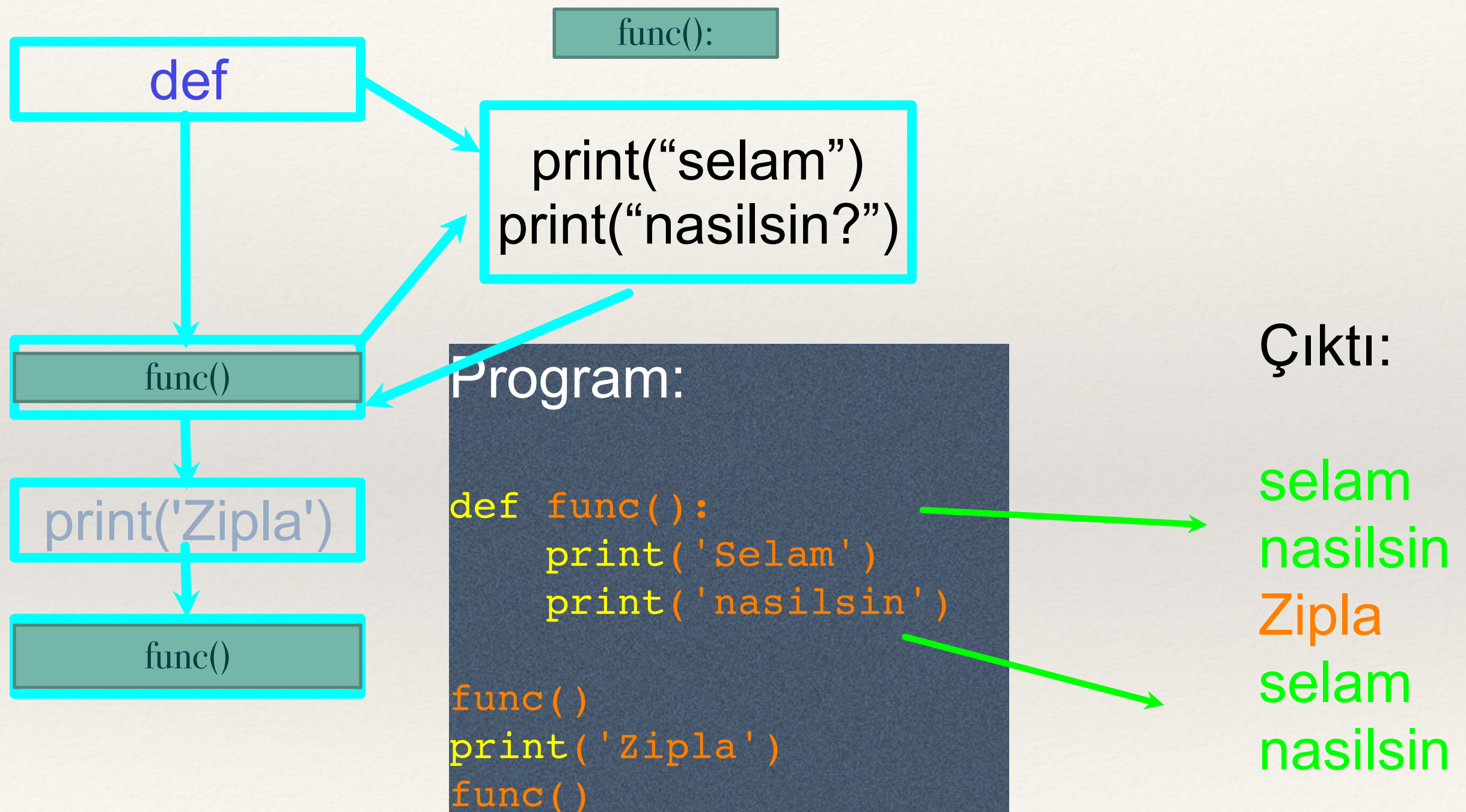

Python ile Programlamaya Giriş

DERS 9: FONKSİYONLAR

DR. HÜSEYİN BAHTİYAR

Kaydedilen (ve tekrar kullanılan) adımlar



We call these reusable pieces of code “functions”

Python Fonksiyonları

- ❖ Python da iki çeşit fonksiyon tipi bulunmaktadır.
 - ❖ **yerleşik fonksiyonlar** bunlar python'ın içerisinde bulunan hazır fonksiyonlardır.

`print()`, `input()`, `type()`, `float()`.. gibi

- ❖ **Bizim tanımladığımız fonksiyonlar**, sonrasında kullandığımız fonksiyonlar
- ❖ Doğal olarak python içindeki yerleşik fonksiyonlar ile aynı isimde fonksiyon tanımlayamayız.

Fonksiyon Tanımı

- ❖ Python da **fonksiyon**'un tanımı, **argüman**(lar)ı girdi olarak alıp hesaplamalar yapan ve sonucunda bu sonuçları çıktı olarak döndüren tekrar kullanılabilir kodlardır.
- ❖ **fonksiyon** tanımlarken **def** kelimesi ile tanımlarız.
- ❖ **Fonksiyonun** adını kullanarak fonksiyonu çağırırız, parantezlerin içerisine de **argümanları** veririz.

Argüman

big = **max**('Hello world')

Atama

'w'

Sonuç

```
>>> big = max('Hello world')
>>> print(big)
w
>>> tiny = min('Hello world')
>>> print(tiny)

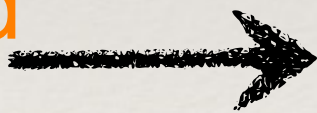
>>>
```

Max Fonksiyonu

```
>>> big = max('Hello world')
>>> print(big)
w
```

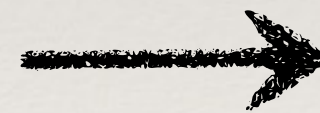
Max fonksiyonu, programın içerisine kaydedilmiş bizim kullandığımız bir fonksiyon. Bu fonksiyon bir **girdi** alıp **çıkı**tı üretiyor.

'Hello world'
(string)



max()
fonksiyonu

Guido yazmış



'w'
(string)

Max Fonksiyonu

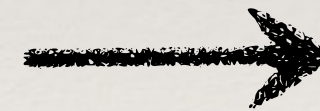
```
>>> big = max('Hello world')
>>> print(big)
w
```

Max fonksiyonu, programın içerisine kaydedilmiş bizim kullandığımız bir fonksiyon. Bu fonksiyon bir **girdi** alıp **çıkı**tı üretiyor.

'Hello world'
(string)



```
def max(inp):
    blah
    blah
    for x in inp:
        blah
        blah
```



'w'
(string)



Guido yazmış

Tip Dönüşümleri

- ❖ Daha önce yaptığımız tip dönüşümü işlemlerinde de fonksiyon kullanıyorduk.

```
>>> print(float(99) / 100)
0.99
>>> i = 42
>>> type(i)
<class 'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> type(f)
<class 'float'>
>>> print(1 + 2 * float(3) / 4 - 5)
-2.5
>>>
```

Kendi fonksiyonumuzu yazmak

- ❖ Yeni bir fonksiyon tanımlarken **def** komutunu kullanıp parantezlerin içerisinde isteğe bağlı parametreler verebiliriz.
- ❖ Fonksiyonun ana kısmını yazarken tab ile indentation yaparız.
- ❖ Bu işlem fonksiyonu **tanımlar** ancak **çalıştırmaz**.

```
def sarki_sozu():  
    print("I'm a lumberjack, and I'm okay.")  
    print('I sleep all night and I work all day.')
```

Kendi fonksiyonumuzu yazmak

```
#!/python3
x = 5
print('Hello')

def sarki_sozu():
    print("I'm a lumberjack, and I'm okay.")
    print('I sleep all night and I work all day.')

print('Yo')
x = x + 2
print(x)
```

Hello
Yo
7

```
sarki_sozu():
    print("I'm a lumberjack, and I'm okay.")
    print('I sleep all night and I work all day.')
```

Tanımlama ve Kullanma

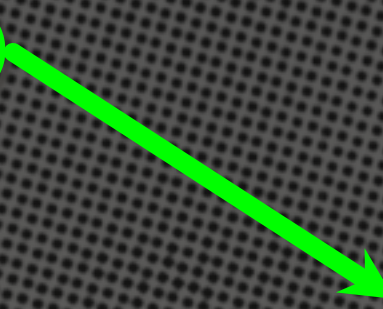
- ❖ Fonksiyonu tanımladıktan sonra, istediğimiz kadar çağırabiliriz.
- ❖ Bu yöntem saklayıp tekrar kullanma yöntemidir.

Kendi fonksiyonumuzu yazmak

```
#!/python3
x = 5
print('Hello')

def sarki_sozu():
    print("I'm a lumberjack, and I'm okay.")
    print('I sleep all night and I work all day.')

print('Yo')
sarki_sozu()
x = x + 2
print(x)
```



Hello
Yo

I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.

Argümanlar

- ❖ Argümanlar fonksiyonu çağırdığımızda fonksiyonun içine girdi olarak verdiğimiz değişkenlerdir.
- ❖ Argümanları kullanmamızın sebebi fonksiyonu çağırdığımızda farklı şekilde çalışmasını sağlamak istememizdir.
- ❖ Fonksiyonun adından sonra koyduğumuz parantezlerin içerisine yazarız.

big = **max**('Hello world')

Argüman


Parametreler

- ❖ Parametreler ise fonksiyonun tanımında kullandığımız değişkenlerdir. Fonksiyonun farklı özelliklerini çağırmak için kullanabiliriz.

```
>>> def selamla(lang):  
...     if lang == 'es':  
...         print('Hola')  
...     elif lang == 'fr':  
...         print('Bonjour')  
...     elif lang == 'en':  
...         print('Hello')  
...     else:  
...         print('Selam')  
...  
>>> selamla('en')  
Hello  
>>> selamla('es')  
Hola  
>>> selamla('fr')  
Bonjour  
>>>
```

Dönüş Değerleri

- ❖ Sıklıkla, fonksiyon argümanları alıp hesaplamalar yapar ve sonucunda bir sonuç değeri döndürür. bunun için **return** komutunu kullanırız.

```
def greet():  
    return "Hello"  
  
print(greet(), "Glenn")  
print(greet(), "Sally")
```

```
Hello Glenn  
Hello Sally
```

Dönüş Değeri

- ◆ Kullanışlı bir fonksiyon sonucunda bir dönüş değeri üretir.
- ◆ return komutu fonksiyonun çalışmasını bitirir ve bulduğu değeri fonksiyonun sonucu olarak geri gönderir.

```
>>> def selamla(lang):  
...     if lang == 'es':  
...         print('Hola')  
...     elif lang == 'fr':  
...         print('Bonjour')  
...     elif lang == 'en':  
...         print('Hello')  
...     else:  
...         print('Selam')  
...  
>>> print(selamla('en'), 'Glenn')  
Hello Glenn  
>>> print(selamla('es'), 'Sally')  
Hola Sally  
>>> print(selamla('fr'), 'Michael')  
Bonjour Michael  
>>>
```

Argümanlar, Parametreler ve Sonuçlar

```
>>> big = max('Hello world')  
>>> print(big)  
w
```

Parametre

Argüman

'Hello world'



```
def max(inp):  
    blah  
    blah  
    for x in inp:  
        blah  
        blah  
    return 'w'
```



'w'

Sonuç

Birden Fazla Parametre/Argüman

- ❖ Fonksiyonda eğer istersek birden fazla parametre tanımlayabiliriz.
- ❖ Böylece basitçe daha fazla argüman ekleyerek fonksiyonu çağırırız.

```
def toplama(a, b):  
    toplam = a + b  
    return toplam
```

```
x = toplama(3, 5)  
print(x)
```

8

Void (kullanışlı olmayan) fonksiyonlar

- ❖ Fonksiyon sonucunda bir değer döndürmüyor ise bu fonksiyona “void” adı verilir.
- ❖ Sonuç döndüren fonksiyonlar kullanışlı fonksiyonlardır.
- ❖ Void fonksiyonları kullanışlı değildir.

Özyineli (recursive) fonksiyonlar

- ❖ Bazı özel durumlarda fonksiyonların kendini tekrar çağırıp aynı işlemi tekrar yapmasını sağlamak isteriz.
- ❖ Bu şekilde yazılan fonksiyonlara özyineli (recursive) fonksiyonlar adı verilir.
- ❖ Özyineli fonksiyon yazarken dikkat etmemiz gereken birkaç önemli özellik vardır.
 - ❖ Başlangıç değeri.
 - ❖ Bitiş değeri
 - ❖ Adım sayısı
- ❖ Yukarıdaki kurallara dikkat ederek tanımlamazsak, fonksiyonumuz yanlış çalışabilir.

Özyineli fonksiyonlar

Örnek

$$3! = 3 * 2 * 1$$
$$n! = n * n-1 * n-2 * \dots 1$$

```
#!/python3
```

```
def faktoriyel(n):  
    if n == 1:  
        return 1  
    else:  
        return n * faktoriyel(n-1)
```

```
sayi=int(input("sayi giriniz: "))  
sonuc=faktoriyel(sayi)  
print(sayi,"faktoriyeli :",sonuc)
```

n değeri için
fonksiyonu başlatıyoruz

n-1 değeri için
fonksiyonu tekrar
çağırıyoruz

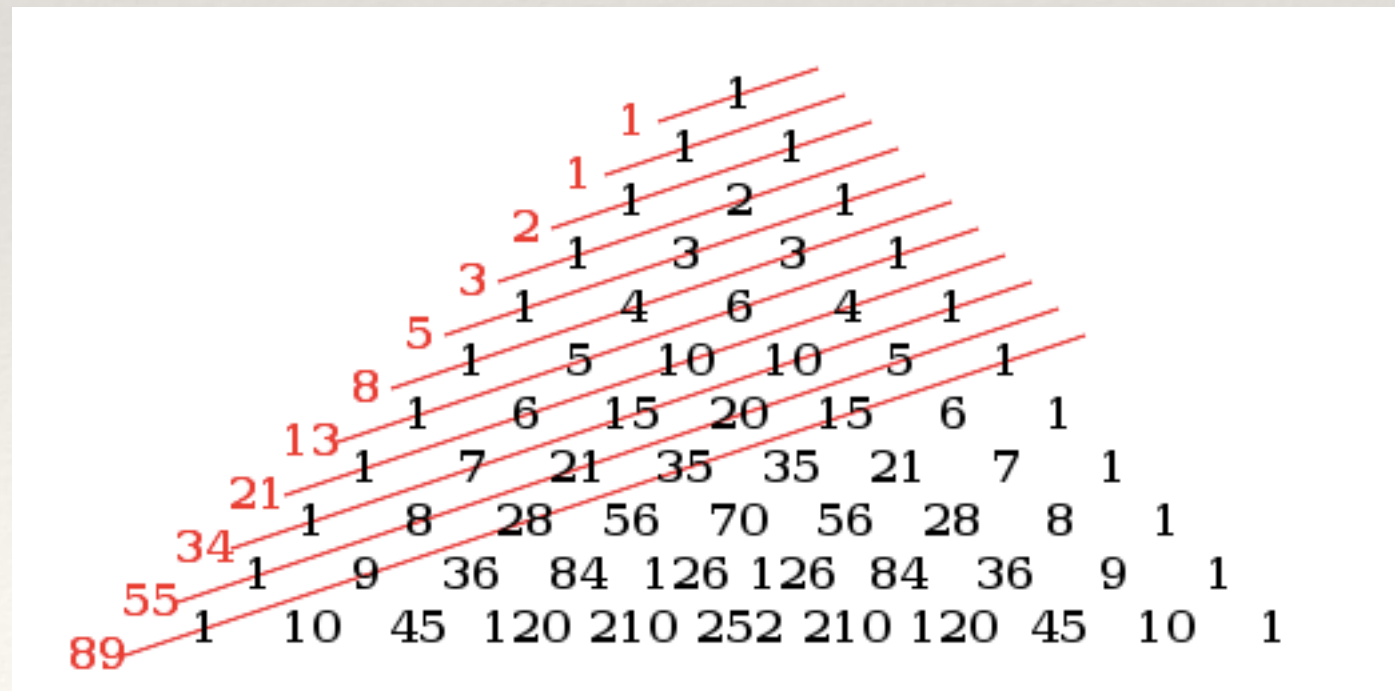
Ne zaman fonksiyon yazmalıyız

- ❖ Programımızı ayrı bölümlere bölmek istersek.
- ❖ Kendimizi tekrar etmemeliyiz, aynı işlemleri kod içinde tekrar tekrar yapmamalıyız.
- ❖ Eğer bir program çok uzun ve kompleks ise, bu durumda mantıklı parçalara bölüp fonksiyonlar şeklinde çağırmak işlem kolaylığı sağlar.

Ödev Matematik

Fibonacci dizisi, her sayının kendinden öncekiyle toplanması sonucu oluşan bir sayı dizisidir. Bu şekilde devam eden bu dizide sayılar birbirleriyle oranlandığında altın oran ortaya çıkar, yani bir sayı kendisinden önceki sayıya bölündüğünde altın orana gittikçe yaklaşan bir dizi elde edilir.

Ödev: Klavyeden girilen sayının Fibonacci dizisinde kaça denk geldiğini veren program yazınız (özyineli fonksiyon ile).



Ödev Diğer

- ❖ Alan isimli bir fonksiyonum olsun bu fonksiyonun a,b isimli iki argümanı olsun.
- ❖ Dikdörgen şeklinde bir yapının alanı $A = a * b$ şeklindedir.
- ❖ Klavyeden girdiğim a ve b sayılarını, Alan fonksiyonuna gönderip dikdörtgenin alanını geri döndüren program yazınız.

Acknowledgements / Contributions

These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and open.umich.edu and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

... Insert new Contributors and Translators here

