Read the Docs Template Documentation

Release 1.0

Read the Docs

Sep 16, 2023

İçindekiler

1 Dikkatli olun **3**

2 Katkıda bulunun **5**

3 Destek **7**

3.1 Python'a Giriş . . . 7
3.2 Python Başlarken . . . 8
3.3 Python Söz Dizimi . . . 9
3.4 Python Değişkenleri . . . 11
3.5 Python Sayılar . . . 12
3.6 Python Tip Dönüşümleri . . . 13
3.7 Python Karakter Dizileri . . . 14
3.8 Python İşleçler(Operatörler) . . . 16
3.9 Python Listeler . . . 18
3.10 Python Demetler . . . 19
3.11 Python Kümeler . . . 20
3.12 Python Sözlükler . . . 21
3.13 Python Koşullar . . . 22
3.14 Python While Döngüleri . . . 24
3.15 Python For Döngüleri . . . 25
3.16 Python Fonksiyonları . . . 27
3.17 Python Dosya Yönetimi . . . 29
3.18 Python Dosya Açma . . . 30
3.19 Python Dosya Oluşturma ve Yazma . . . 31
3.20 Python Dosya Silme . . . 32
3.21 Authors . . . 33
3.22 Contributing . . . 33
3.23 History . . . 33
3.24 Installation . . . 33
3.25 Python Öğren Belgelerine Hoşgeldiniz! . . . 33
3.26 Usage . . . 34

4 Indices and tables **35**

python-ogren projesi https://www.w3schools.com/python/ adresindeki anlatımların çevirilerini içerir. Python hakkında ek bir Türkçe kaynak olmayı hedefler.

Belgelendirmeye erişmek için: http://python-ogren.readthedocs.io

CHAPTER 1

Dikkatli olun

- Çeviriler henüz düzenleme aşamasındadır.
- Program kodlarının Türkçeleştirilmesi sırasında hatalar olabilir.
- Çeviriler henüz üçüncü bir göz ile inclenmediği için karmaşık gelebilir.

CHAPTER 2

Katkıda bulunun

- Düzenleme İsteği: https://github.com/ozgurturkiye/python-ogren/pulls
- Kaynak Kod: https://github.com/ozgurturkiye/python-ogren
- Belgenin özgün kaynağı: https://www.w3schools.com/python/

CHAPTER 3

Destek

Eğer katkıda bulunmak isterseniz lütfen bize bildirin. İletişim için: ozgurturkiye@gmail.com veya hurolyalcin@gmail.com

3.1 Python'a Giriş

3.1.1 Python Nedir?

Python popüler bir programlama dilidir ve 1991 yılında "Guido van Rossum" tarafından geliştirilmiştir.

Ne için kullanılır:

- sunucu taraflı web geliştirme
- yazılım geliştirme
- matematik
- sistem betikleri yazma

3.1.2 Python neler yapabilir?

- Python, web uygulamaları oluşturmak için bir sunucuda kullanılabilir.
- Python, iş akışları oluşturmak için uygulamaların yanında kullanılabilir.
- Python veritabanı sistemlerine bağlanabilir. Ayrıca dosyaları okuyabilir ve değiştirebilir.
- Python büyük verileri işlemek ve karmaşık matematik yapmak için kullanılabilir.
- Python hızlı prototipleme veya hazır ürün geliştirme için kullanılabilir.

3.1.3 Peki Neden Python?

- Python farklı platformlarda çalışır (Windows, Mac, Linux, Raspberry Pi, vb.).
- Python, İngilizce'ye benzer basit bir söz dizimine sahiptir.
- Python, geliştiricilerin, diğer programlama dillerinden daha kısa programlar yazmasına olanak veren bir söz dizimine sahiptir.
- Python, bir interpreter(yorumlayıcı) sistemiyle çalışır, yani kod yazıldığı zaman yürütülebilir. Bu, prototip oluşturmanın çok hızlı olabileceği anlamına gelir.
- Python kodları prosedürel, nesne yönelimli veya fonksiyonel bir şekilde yazılabilir.

3.1.4 Ek Bilgiler

- Python'un en yeni ve ana sürümü, bu derste kullanacağımız Python 3'tür. Bununla birlikte, Python 2 kullanılmaya devam etse bile geçerli bir sebebiniz yoksa yeni sürümü kullanın.
- Buradaki anlatımlarda Python basit bir metin editöründe yazılacaktır. Python'u, daha gelişmiş araçlar kullanarak yazmak isterseniz Thonny, Pycharm, Netbeans veya Eclipse gibi Entegre Geliştirme Ortamları kullanarak da yazmak mümkündür.

3.1.5 Python söz diziminin diğer programlama dilleriyle karşılaştırılması

- Python'ın tasarımında okunabilirlik ön plandadır ve matematikten etkilenen İngilizce diliyle bazı benzerliklere sahiptir.
- Python, genellikle noktalı virgül veya parantez kullanan diğer programlama dillerinin aksine, bir komutu tamamlamak için yeni satırlar kullanır.
- Python, kapsamı tanımlamak için girintiler kullanır; Tıpki döngüler, fonksiyonlar ve sınıfların kapsamı gibi. Diğer programlama dilleri genellikle bu amaç için süslü parantezler kullanır.

3.2 Python Başlarken

3.2.1 Python Yükleme

Birçok PC ve MAC bilgisayarda; Python zaten yüklü gelecektir. Windows yüklü bir PC'de Python yüklü olup olmadığını kontrol etmek için; arama çubuğunda Python'ı aratın veya aşağıdaki komutu komut satırında yürütün. (cmd.exe)

Örnek:

```
C:\Users\Your Name(Kullanıcı Adı)\python --version
```

Linux yada MAC yüklü bilgisayarınızda Python yüklü olup olmadığını kontrol etmek için; Linux'de komut satırına veya MAC'de Terminal'e şu komutu yazın.:

```
python --version
```

Eğer bilgisayarınızda pyhon yüklü değilse; linkteki siteden özgürce indirebilirsiniz. https://www.python.org/

3.2.2 Python Hızlı Başlangıç

Python yorumlanan bir programlama dilidir. Yani; geliştirici olarak siz .py uzantılı Python kodlarınızı bir metin düzenleyicide yazar ve çalıştırmak için Python yorumlayıcısına aktarırsınız.

Bir Python dosyasını yürütmenin yolu aşağıdaki gibi bir komutu komut satırında çalıştırmaktan ibarettir.:

```
C:\Users\Your Name(Kullanıcı Adı)\python merhaba_dunya.py
```

Hadi herhangi bir metin düzenleyicide "merhaba_dunya.py" adlı ilk Python dosyamızı yazalım.

merhaba_dunya.py:

```
print("Merhaba, Dünya!")
```

İşte bu kadar basit. Dosyanızı kaydedin. Komut satırını açıp dosyanızı kaydettiğiniz dizine girin ve çalıştırın:

```
C:\Users\Your Name(Kullanıcı Adı)\python merhaba_dunya.py
```

Şöyle bir çıktı göreceksiniz:

```
Merhaba, Dünya!
```

Tebrikler, ilk Python programınızı yazdınız ve çalıştırdınız.

3.2.3 Python Komut Satırı

Python'da kısa bir kodu test etmek için bazen kodu bir dosyaya yazmamak en hızlı ve kolay yoldur. Bu, Python aynı zamanda bir komut satırı olarak çalıştırılabildiği için mümkündür.

Windows, Mac veya Linux komut satırında aşağıdakileri yazın:

```
C:\Users\Your Name(Kullanıcı Adı)\python3
```

Burada, daha önce gelen merhaba dünya örneğimiz de dahil herhangi bir python kodu yazabilirsiniz:

```
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, World!")
```

Komut satırında "Hello, World!" yazacak:

```
Hello, World!
```

Python komut satırı ile işiniz bitince, python komut satırı arabiriminden çıkmak için GNU/Linux terminalde "ctrl-D" kısayolu kullanabilir veya aşağıdakini yazabilirsiniz:

```
exit()
```

3.3 Python Söz Dizimi

3.3.1 Python Komutlarını Çalıştırmak

Bir önceki sayfada öğrendiğimiz gibi, Python komutları doğrudan Komut Satırında yazılarak çalıştırılabilir:

```
>>> print("Hello, World!")
Hello, World!
```

Ya da sunucuda .py uzantılı bir python dosyası oluşturup bu dosyayı Komut Satırında çalıştırarak:

```
C:\Users\Your-Name>python myfile.py
```

3.3.2 Python Girintiler

Diğer programlama dillerinde, koddaki girinti sadece okunabilirlik içindir fakat Python'da girinti çok önemlidir.

Python bir kod bloğunu belirtmek için girinti kullanır.

Örnek:

```
if 5 > 2:
  print("Five is greater than two!")
```

Girintiyi atlarsanız Python size bir hata verecektir:

Örnek:

```
if 5 > 2:
print("Five is greater than two!")
```

3.3.3 Yorumlar

Python, kod içinde yorum ekleme yeteneğine sahiptir.

Yorumlar # ile başlar ve bundan sonra Python satırın geri kalanını yorum olarak verir:

Örnek:

Python yorum satırı:

```
#This is a comment.
print("Hello, World!")
```

3.3.4 Docstrings

Python ayrıca belge dizisi olarak adlandırılan genişletilmiş belge kapasitesine sahiptir. Docstrings bir satır veya çok satırlı olabilir. Python, docstring'in başında ve sonunda üçlü tırnak kullanır:

Örnek:

Docstrings'ler yorum yazmak için de kullanılabilir:

```
"""This is a
multiline docstring."""
print("Hello, World!")
```

3.4 Python Değişkenleri

3.4.1 Değişken Tanımlama

Diğer programlama dillerinden farklı olarak, Python'ın bir değişken bildirme komutu yoktur.

İlk değer atadığınız anda değişken tanımlanır. Örnek:

```
x = 5
y = "Özgür"
print(x)
print(y)
```

Değişkenlerin herhangi bir tiple bildirilmesi gerekmez ve ilk değer atandıktan sonra bile tipi değiştirebilir.

Örnek:

```
x = 4 # x tam sayı tipinde
x = "Fikri" # x artık string tipinde
print(x)
```

3.4.2 Değişken Adları

Bir değişken kısa bir isme (x ve y gibi) veya daha açıklayıcı bir isme sahip olabilir (yaş, carname, toplam_hacim)

Değişken Adlandırma Kuralları:

- Değişken adı bir harfle yada altçizgi karakteriyle başlamalıdır.
- Değişken adı sayı ile başlayamaz.
- Değişken adları alfanümerik karakterler ve altçizgi içerebilir. (A-z, 0-9, and _)
- Değişken adları büyük/küçük harf duyarlıdır.(yaş, Yaş ve YAŞ üç farklı değişkendir.)

Değişken adlarının büyük/küçük harf duyarlı olduğunu unutmayın.

3.4.3 Değişkenlerin Çıktısı

Python print ifadesi genellikle değişkenlerin çıktısını almak için kullanılır. Hem metni hem de bir değişkeni birleştirmek için Python, + karakterini kullanır:

Örnek:

```
x = "awesome"
print("Python is " + x)
```

Başka bir değişkene başka bir değişken eklemek için + operatörünü kullanabilirsiniz.

Örnek:

```
x = "awesome"
y = "Peter is " + x
print(x)
```

Sayılar için + karakteri matematiksel bir operatör olarak çalışır:

Örnek:

```
x = 5
y = 10
print(x + y)
```

Eğer bir karakter dizisini(str) ve sayıyı(int) + operatörü ile birleştirmeye çalışırsanız; Python karakter dizisi ve sayıyı birleştiremeyeceğine dair hata verecektir.

Örnek:

```
x = 5
y = "John"
print(x + y)
```

3.5 Python Sayılar

3.5.1 Python Sayılar

Python'da üç sayısal tür vardır:

- int (integer)
- float (kayan noktalı)
- complex (karmaşık)

Sayısal türdeki değişkenler, onlara bir değer atadığınızda oluşturulur:

Python'daki herhangi bir nesnenin türünü doğrulamak için, *type()* fonksiyonunu kullanın:

Örnek:

```
print(type(x))
print(type(y))
print(type(z))
```

3.5.2 int

integer veya tamsayı, sınırsız uzunlukta, ondalıksız, tam sayı, pozitif veya negatif sayılardır.

Örnek:

Integers:

```
x = 1
y = 35656222554887711
z = -3255522

print(type(x))
print(type(y))
print(type(z))
```

3.5.3 float

Kayan nokta veya "kayan nokta sayısı", bir veya daha fazla ondalık basamak içeren pozitif veya negatif sayılardır.

Örnek

Floats:

```
x = 1.10
y = 1.0
z = -35.59

print(type(x))
print(type(y))
print(type(z))
```

Float ayrıca, 10'un katlarını göstermek için bir "e" ile bilimsel sayılar da olabilir.

Örnek

Floats:

```
x = 35e3
y = 12E4
z = -87.7e100

print(type(x))
print(type(y))
print(type(z))
```

3.5.4 complex

Karmaşık sayılarda sanal kısım "j" kullanılarak yazılır:

Örnek:

Complex:

```
x = 3+5j
y = 5j
z = -5j

print(type(x))
print(type(y))
print(type(z))
```

3.6 Python Tip Dönüşümleri

3.6.1 Değişken Tipi Tanımlama

Bir değişkeni belli bir türde tanımlamak istediğiniz zamanlar olabilir. Bu tip dönüşümü ile yapılabilir. Python nesne yönelimli bir dildir ve veri tiplerini tanımlamak için sınıfları kullanır.

Python'da tip dönüşümü, tip dönüştürme fonksiyonları kullanılarak yapılır:

- *int()* - bir tam sayıdan, bir kayan noktalı sayıdan veya değeri tam sayı olan bir karakter dizisinden tam sayı oluşturur.
- *float()* - bir tamsayıdan, bir kayan noktalı sayıdan ve değeri tam sayı veya kayan noktalı sayı olan karakter dizisini kayan noktalı(float) sayıya dönüştürür.

- *str()* - birçok veri tipini karakter dizisine dönüştürür.

Örnek:

Integers:

```
x = int(1)   # x değeri 1 olur
y = int(2.8) # y değeri 2 olur
z = int("3") # z değeri 3 olur
```

Örnek

Floats:

```
x = float(1)     # x değeri 1.0 olur
y = float(2.8)   # y değeri 2.8 olur
z = float("3")   # z değeri 3.0 olur
w = float("4.2") # w değeri 4. olur
```

Örnek

Strings:

```
x = str("s1") # x değeri 's1' olur
y = str(2)    # y değeri '2' olur
z = str(3.0)  # z değeri '3.0' olur
```

3.7 Python Karakter Dizileri

3.7.1 Karakter Dizisi Tanımlama

Python'daki karakter dizileri, tek tırnak işaretleri veya çift tırnak işaretleri ile çevrelenir.

'hello' ile *"hello"* aynıdır.

Karakter dizileri print fonksiyonu kullanılarak ekrana basılabilir. Örnek olarak: *print("hello").*

Diğer birçok popüler programlama dilleri gibi, Python karakter dizileri de unicode karakterleri temsil eden bayt dizileridir. Bununla birlikte, Python'un bir karakter veri türü yoktur, tek bir karakter sadece 1 uzunluğunda bir dizedir. Dizenin elemanlarına erişmek için köşeli parantez kullanılabilir.

Örnek:

Birinci pozisyondaki karakteri alır:

```
a = "hello"
print(a[1])
```

Örnek

Karakter dizisinin belli bir bölümünü almak. 2. karakterden başlayarak 5. karaktere kadar alır:

```
b = "world"
print(b[2:5])
```

Örnek

The *strip()* method removes any whitespace from the beginning or the end: *strip()* metodu, herhangi bir boşluğu başlangıç veya bitişten kaldırır:

```
a = " Hello, World! "
print(a.strip()) # returns "Hello, World!"
```

Örnek

Karakter dizisinin boyutunu verir:

```
a = "Hello, World!"
print(len(a))
```

Örnek

lower() metodu küçük harfe döndürür:

```
a = "Hello, World!"
print(a.lower())
```

Örnek *upper()* metodu büyük harfe döndürür:

```
a = "Hello, World!"
print(a.upper())
```

Örnek

replace() methodu bir karakteri başka bir karakterle değiştirir:

```
a = "Hello, World!"
print(a.replace("H", "J"))
```

Örnek

split() methodu karakter dizisini verilen ayırıyıcıya göre böler:

```
a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']
```

3.7.2 Komut Satırı Karakter Dizisi girişi

Python komut satırı girişi için izin verir. Bu, kullanıcıdan girdi isteyebileceğimiz anlamına gelir. Aşağıdaki örnek *input()* fonksiyonunu kullanarak kullanıcının adını sorar, daha sonra *print()* fonksiyonunu kullanarak, adı ekrana yazdırır:

Örnek

demo_string_input.py:

```
print("Enter your name:")
x = input()
print("Hello, " + x)
```

Bu dosyayı *demo_string_input.py* olarak kaydedin ve komut satırından yükleyin:

```
C:\Users\Your Name>python demo_string_input.py
```

Programımız kullanıcıyı bir karakter dizisi girişine yönlendirecek:

```
Enter your name:
```

Kullanıcı şimdi bir isim giriyor:

```
Linus
```

Ardından, program küçük bir mesajla ekrana girilen değeri yazdırır:

```
Hello, Linus
```

3.8 Python İşleçler(Operatörler)

Bu konuda İngilizce operatör kavramı yerine Türkçeye daha uygun olan işleç kavramı kullanacağız.

3.8.1 Python İşleçler

İşleçler değişkenler ve değerler üzerinde işlem yapmak için kullanılır. Python, işleçleri aşağıdaki gruplara ayırır:

- Aritmetik işleçler
- Atama işleçleri
- Karşılaştırma işleçleri
- Mantıksal işleçler
- Kimlik işleçleri
- Üyelik işleçleri
- Bit düzeyi işleçler

3.8.2 Python Aritmetik İşleçleri

Aritmetik işleçler, matematiksel işlemleri gerçekleştirmek için sayısal değerlerle kullanılır:

İşleç	İsim	Örnek
+	Toplama	x + y
−	Çıkarma	x - y
*	Çarpma	x * y
/	Bölme	x / y
%	Mod alma	x % y
**	Kuvvet alma	x ** y
//	Taban bölme	x // y

3.8.3 Python Atama İşleçleri

Atama işleçleri değişkenlere değer atamak için kullanılır.

Not: “Örnek-1” ve “Örnek-2” aynı işlemi yapar.

İşleç	Örnek-1	Örnek-2
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

3.8.4 Python Karşılaştırma İşleçleri

Karşılaştırma operatörleri iki değeri karşılaştırmak için kullanılır:

İşleç	İsim	Örnek
==	Equal	x = y
!=	Not equal	x != y
<>	Not equal	x <> y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

3.8.5 Python Mantıksal İşleçler

Mantıksal işleçler koşullu ifadeleri birleştirmek için kullanılır:

İleç	Açıklama	Örnek
`and`	Her iki ifade doğruysa, True döndürür False	x < 5 and x < 10
`or`	İfadelerden biri doğruysa True döndürür False	x < 5 or x < 4
`not`	Sonucu tersine döndürür, sonuç doğru ise False döndürür	not(x < 5 and x < 10)

3.8.6 Python Kimlik İşleçleri

Kimlik operatörleri nesneleri karşılaştırmak için kullanılırlar, nesnelerin içeriğinin eşit olmasına bakmaz, tam olarak aynı bellek adresinde ki aynı nesne olmasına bakar: Karışık geliyorsa detaylı anlatım için: https://belgeler.yazbel.com/python-istihza/islecler.html#kimlik-islecleri

İşleç	Açıklama	Örnek
`is`	Her iki değişkenin aynı nesne olması durumunda True döndürür	x is y
`is not`	Her iki değişken aynı nesne ise, yanlış döndürür	x is not y

3.8.7 Python Üyelik İşleçleri

Üyelik işleçleri, bir nesnenin bir nesnede sunulup sunulmadığını test etmek için kullanılır:

İşleç	Açıklama	Örnek
`in`	Nesnede belirtilen değere sahip bir dizi varsa True döndürür	x in y
`not in`	Nesnede belirtilen değere sahip bir dizi varsa False döndürür	x not in y

3.8.8 Python Bit Düzeyi İşleçler

Mantıksal işleçler koşullu ifadeleri birleştirmek için kullanılır:

İşleç	İsim Açıklama	
`&`	AND	Her iki bit 1 ise, her biti 1'e ayarlar x in y
`|`	OR	İki bitten biri 1 ise her bit 1'i ayarlar x not in y
`^`	XOR	İki bitten sadece biri 1 ise her biti 1'e ayarlar
`~`	NOT	Tüm bitleri ters çevirir
`<<`	Sıfır doldurmalı sola kaydırma	Sıfırları sağdan içeri doğru iterek sola kaydırır
`>>`	Signed right shift	En soldaki bitin kopyalarını sola doğru iterek sağa kaydırır ve en sağdaki bitlerin düşmesini sağlar

3.9 Python Listeler

3.9.1 Python Kolleksiyonları(Diziler)

Python programlama dilinde dört koleksiyon veri türü vardır:

- **List** sıralanabilen ve değiştirilebilen bir veri türüdür. Yinelenen üyelere izin verir.
- **Tuple**, sıralanabilen ve değiştirilemeyen bir veri türüdür. Yinelenen üyelere izin verir.
- **Set**, sırasız ve işaretsiz olan bir veri türüdür. Yinelenen üye yoktur.
- **Dictionary**, sırasız, değiştirilebilir ve indeksli bir veri türüdür. Yinelenen üye yoktur.

Bir koleksiyon türü seçerken, bu türün özelliklerini anlamak yararlıdır. Belirli bir veri seti için doğru tipin seçilmesi, anlamın korunmasını ifade edebilir ve bu verimlilikte veya güvenlikte bir artış anlamına gelebilir.

3.9.2 List

Liste, sıralanabilen ve değiştirilebilen bir koleksiyondur. Python listelerinde köşeli parantez ile yazılır.

Örnek

Create a List:

```
ornek_liste = ["apple", "banana", "cherry"]
print(ornek_liste)
```

Örnek

Change the second item:

```
ornek_liste = ["apple", "banana", "cherry"]
ornek_liste[1] = "blackcurrant"
print(ornek_liste)
```

3.9.3 list() - Liste Oluşturucu

Liste yapmak için `list()` fonksiyonunu kullanmak da mümkündür. Listeye bir öğe eklemek için `append()` nesne metodunu kullanın. Belirli bir öğeyi kaldırmak için `remove()` metodunu kullanın. `len()` fonksiyonu, listenin uzunluğunu döndürür.

Örnek

`list()` - Liste oluşturucusunu liste oluşturmak için kullanmak:

```
ornek_liste = list(("apple", "banana", "cherry")) # note the double round-brackets
print(ornek_liste)
```

Örnek

`append()` metodu ile listeye üye ekleme:

```
ornek_liste = list(("apple", "banana", "cherry"))
ornek_liste.append("damson")
print(ornek_liste)
```

Örnek

`remove()` metodu ile listeden üye silme:

```
ornek_liste = list(("apple", "banana", "cherry"))
ornek_liste.remove("banana")
print(ornek_liste)
```

Örnek

`len()` metodu ile listenin üye sayısını bulma:

```
ornek_liste = list(("apple", "banana", "cherry"))
print(len(ornek_liste))
```

3.10 Python Demetler

3.10.1 Demet

Demet(tuple), sıralanabilen ve değiştirilemeyen bir veri türüdür. Python demetlerde yuvarlak köşeli ayraçlar bulunur.

Örnek

Bir demet oluşturmak:

```
ornek_demet = ("elma", "muz", "çilek")
print(ornek_demet)
```

Örnek

Birinci sıradaki nesneyi alma:

```
ornek_demet = ("elma", "muz", "çilek")
print(ornek_demet[1])
```

Örnek

Bir demetteki değerleri değiştiremezsiniz:

```
ornek_demet = ("elma", "muz", "çilek")
ornek_demet[1] = "blackcurrant" # değişebilirliği test et
print(ornek_demet)
```

3.10.2 tuple() - Demet Oluşturucu

Demet oluşturmak için `tuple()` yapıcısını kullanmak da mümkündür. `len()` fonksiyonu, demet uzunluğunu döndürür.

Örnek

`tuple()` fonksiyonunu kullanarak bir demet oluşturmak:

```
ornek_demet = tuple(("elma", "muz", "çilek")) # iki parantez olduğuna dikkat edin
print(ornek_demet)
```

Örnek

`len()` metodu demetin içindeki nesne sayısını döndürür:

```
ornek_demet = tuple(("elma", "muz", "çilek"))
print(len(ornek_demet))
```

Not: Bir demetteki öğeleri kaldıramazsınız. Demetler değiştirilemez(immutable) veri türleridir.

3.11 Python Kümeler

3.11.1 Küme

Bir küme, sırasız ve işaretsiz olan bir veri türüdür. Python kümelerinde küme parantezleri ile yazılır.

Örnek

Küme oluşturma:

```
ornek_kume = {"apple", "banana"}
print(ornek_kume)
```

Not: Küme listesi sırasızdır, bu yüzden öğeler rastgele sırayla görünecektir.

3.11.2 set() - Küme Oluşturucu

Küme oluşturmak için `set()` yapıcısını kullanmak da mümkündür. Bir öğe eklemek için `add()` nesne metodunu ve öğeden bir öğeyi kaldırmak için `remove()` nesne metodunu kullanabilirsiniz. `len()` fonksiyonu, kümenin boyutunu döndürür.

Örnek

`set()` oluşturucusunu kullanarak küme oluşturma:

```
ornek_kume = set(("apple", "banana", "cherry")) # çift paranteze dikkat edin
print(ornek_kume)
```

Örnek

`add()` methodu kullarak kümeye öğe ekleme:

```
ornek_kume = set(("apple", "banana", "cherry"))
ornek_kume.add("damson")
print(ornek_kume)
```

Örnek

`remove()` methodu kullanarak kümeden öğe silme:

```
ornek_kume = set(("apple", "banana", "cherry"))
ornek_kume.remove("banana")
print(ornek_kume)
```

Örnek

`len()` fonksiyonunu kullanarak kümedeki öğe sayısını alma:

```
ornek_kume = set(("apple", "banana", "cherry"))
print(len(ornek_kume))
```

3.12 Python Sözlükler

3.12.1 Sözlük

Sözlük, sırasız, değiştirilebilir ve indeksli bir koleksiyondur. Python sözlük veri türünü tanımlarken küme parantezleri kullanılır ve bunların anahtarları ve değerleri vardır.

Örnek

Bir sözlük oluşturma:

```
sözlük = {
  "apple": "green",
  "banana": "yellow",
  "cherry": "red"
}
print(sözlük)
```

Örnek

elma rengini “yesil” olarak değiştirme:

```
sözlük = {
  "elma": "yesil",
  "banana": "yellow",
  "cherry": "red"
}
sözlük["elma"] = "yesil"
print(sözlük)
```

3.12.2 dict() - Sözlük Oluşturucu

Sözlük oluşturmak için `dict()` yapıcısını kullanmak da mümkündür.

Örnek:

```
ornek_sozluk = dict(apple="green", banana="yellow", cherry="red")
# sözlük anahtarlarının karakter dizisi olmadığına dikkat edin
# atama için ``:`` yerine ``=`` kullanıldığına dikkat edin
print(ornek_sozluk)
```

3.12.3 Öğe Ekleme

Sözlüğe bir öğe eklemek, yeni bir indeks anahtarı kullanarak ve buna değer atayarak yapılır:

Örnek:

```
sözlük = dict(apple="green", banana="yellow", cherry="red")
sözlük["damson"] = "purple"
print(sözlük)
```

3.12.4 Öğe Silme

Bir sözlük öğesini kaldırmak python'da `del()` fonksiyonu kullanılarak yapılmalıdır:

Örnek:

```
sözlük = dict(apple="green", banana="yellow", cherry="red")
del(sözlük["banana"]
print(sözlük)
```

3.12.5 Bir Sözlüğün Uzunluğunu Alma

`len()` fonksiyonu, sözlüğün öğe sayısını döndürür:

Örnek:

```
sözlük = dict(apple="green", banana="yellow", cherry="red")
print(len(sözlük))
```

3.13 Python Koşullar

3.13.1 Python Koşulları ve If Deyimleri

Python, matematikteki genel mantıksal koşulları destekler:

- Eşit: `a == b`
- Eşit değil: `a != b`
- Küçüktür: `a < b`
- Küçük eşit: `a <= b`

- Büyüktür: `a > b`
- Büyük eşit : `a >= b`

Bu koşullar, genellikle "deyimler" ve "döngüler"de olmak üzere çeşitli şekillerde kullanılabilir. Bir "If deyimi", `if` anahtar kelimesi kullanarak yazılır.

Örnek

If deyimi:

```
a = 33
b = 200
if b > a: print("b büyüktür a")
```

Bu örnekte, `b`'nin `a`'dan büyük olup olmadığını test etmek için if ifadesinin bir parçası olarak kullanılan iki ve değişkeni vardır. `a = 33` olduğu ve `b = 200` olduğu için, 200'ün 33'den büyük olduğunu biliyoruz ve bu nedenle "b'nin büyük olduğunu" gösterecek şekilde yazdırıyoruz.

3.13.2 Girintileme

Python, koddaki kapsamı tanımlamak için boşlukları kullanarak girintiye dayanır. Diğer programlama dilleri genellikle bu amaç için süslü parantezler kullanır.

Örnek

Yeni satırlardaki ifadeler, girintileri kullanmalıdır:

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

Örnek

If deyimi girinti olmadan:

```
a = 33
b = 200
if b > a:
print("b is greater than a") # Bir hata mesajı alırsınız
```

3.13.3 Elif

`elif` anahtar sözcüğü "önceki koşullar doğru değilse, o zaman bu koşulu kontrol et" demenin pythonca yoludur.

Örnek:

```
a = 33
b = 33
if b > a:
  print("b büyüktür a")
elif a == b:
  print("a ve b eşittir")
```

Bu örnekte `a`, `b`'ye eşittir, bu yüzden ilk koşul doğru değildir, ancak `elif` koşulu doğrudur, bu yüzden ekrana "a ve b eşittir" yazılır.

3.13.4 Else

`else` anahtar kelimesi, önceki koşullar tarafından yakalanmayan her şeyi yakalar.

Örnek:

```
a = 200
b = 33
if b > a:
  print("b büyüktür a")
elif a == b:
  print("a eşittir b")
else:
  print("a büyüktür b")
```

Bu örnekte `a`, `b`'den daha büyüktür, bu yüzden ilk koşul doğru değildir, aynı zamanda *elif* koşulu da doğru değildir, bu yüzden `else` koşuluna gidip "a büyüktür b " yazısını ekrana basıyoruz.

3.14 Python While Döngüleri

3.14.1 Python Döngüleri

Python'un iki ilkel döngü komutu vardır:

- `while` döngüsü
- `for` döngüsü

3.14.2 while Döngüsü

`while` döngüsü ile bir koşul doğru olduğu sürece bir dizi ifade çalıştırabiliriz.

Örnek:

i 6'dan küçük olduğum sürece ekrana i değerini yazdır:

```
i = 1
while i < 6:
  print(i)
  i += 1
```

Note: `i` değerini artırmayı unutmayın, aksi takdirde döngü sonsuza kadar devam edecektir.

`while` döngüsünde, ilgili değişkenlerin hazır olmasını gerektirir, bu örnekte, 1 olarak belirlediğimiz bir indeksleme değişkenini tanımlamamız gerekir.

3.14.3 break Deyimi

`break` deyimi ile while durumu doğru olsa bile döngüyü durdurabiliriz:

Örnek:

i değeri 3 olduğunda döngüden çık:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

3.14.4 continue Deyimi

`continue` deyimi ile mevcut yinelemeyi durdurabilir ve bir sonraki ile devam edebiliriz:

Örnek:

i değeri 3 olduğunda bir sonraki yenilemeye geçer:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

3.15 Python For Döngüleri

3.15.1 Python For Döngüleri

Bir `for` döngüsü, bir dizi üzerinde yineleme yapmak için kullanılır (yani, bir liste, bir demet veya bir dize).

Bu `for` kullanımı, diğer programlama dillerine göre daha az anahtar sözcük kullanır ve diğer nesne yönelimli programlama dillerinde bulunan bir yineleyici yöntemi gibi çalışır.

`for` döngüsü ile bir dizi deyim, liste, demet, küme vb. nesneyi kullanabiliriz.

Örnek

Meyve listesinde ki her bir meyveyi yazdırın:

```
meyveler = ["apple", "banana", "cherry"]
for meyve in meyveler:
  print(meyve)
```

For döngüsünün kendisi buna izin verdiği için for döngüsü önceden ayarlanacak bir indeksleme değişkeni gerektirmez.

3.15.2 break Deyimi

`break` deyimiyle, tüm öğeler arasında döngü oluşturmadan önce döngüyü durdurabiliriz:

Örnek:

i değeri 3 olduğu zaman döngüden çık:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

3.15.3 continue Deyimi

`continue` ifadesiyle, döngüdeki mevcut yinelemeyi durdurabilir ve bir sonraki ile devam edebiliriz:

Örnek

"banana"yı yazdırmaz:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

3.15.4 range() fonksiyonu

Belirli bir sayıda kodla döngü yapmak için range() fonksiyonunu kullanabiliriz, `range()` fonksiyonu 0'dan başlayan ve 1'er artan (varsayılan olarak) ve belirtilen sayıda biten bir sayı dizisi döndürür.

Örnek:

range() fonksiyonu kullanımı:

```
for x in range(6):
  print(x)
```

`range(6)`'nın 0 - 6 değerleri değil, 0 - 5 değerleri olduğunu unutmayın.

`range()` fonksiyonu varsayılan başlangıç değeri olarak olarak 0'dır, ancak başlangıç değerini bir parametre ekleyerek belirtmek mümkündür: `range(2, 6)`, yani 2'den 6'ya kadar olan değerler anlamına gelir (ancak 6'yı içermez):

Örnek

Başlangıç parametresi kullanma:

```
for x in range(2, 6):
  print(x)
```

`range()` fonksiyonu varsayılan olarak diziyi 1 artırır, ancak üçüncü bir parametre ekleyerek artış değerini belirtmek mümkündür: `range(2, 30, 3):`

Örnek

Diziyi 3'er artırır (varsayılan 1'dir):

```
for x in range(2, 30, 3):
  print(x)
```

3.15.5 Özyineleme

Python aynı zamanda fonksiyon tekrarlamasını da kabul eder, bu da tanımlanmış bir fonksiyonun kendisini çağırabileceği anlamına gelir.

Özyineleme, ortak bir matematik ve programlama kavramdır. Bu, bir fonksiyonun kendisini çağırdığı anlamına gelir.

Program geliştiricilerin, hiçbir zaman sonlanmayan veya fazla miktarda bellek ya da işlemci gücü kullanan bir fonksiyon yazması oldukça kolay olduğundan, özyinelemi kullanırken çok dikkatli olmalıdır. Ancak, doğru bir şekilde yazıldığında özyineleme, programlamaya çok verimli ve matematiksel olarak zarif bir yaklaşım olabilir.

Bu örnekte, `tri_recursion()`, kendisini çağırmak için tanımladığımız bir fonksiyondur ("özyineleme"). `k` değişkenini veri olarak kullanıyoruz ve her tekrar ettiğinde (-1) azalır. Durum, koşul 0'dan büyük olmadığı zaman sona erer (diğer bir deyişle 0 olduğunda).

Yeni bir geliştiriciler için, kodun tam olarak nasıl çalıştığını anlamak biraz zaman alacaktır, anlamanın en iyi yolu test etmek ve kodu değiştirerek neler olacağına bakmak olduğunu söyleyebiliriz.

Örnek

Öz yinelemeli örnek:

```
def tri_recursion(k):
  if(k>0):
    result = k+tri_recursion(k-1)
    print(result)
  else:
    result = 0
  return result

print("\n\nÖzyineleme örnek sonuçlar")
tri_recursion(6)
```

3.16 Python Fonksiyonları

Bir fonksiyon, yalnızca çağrıldığında çalışan bir kod bloğudur.

Parametreler olarak bilinen verileri bir fonksiyona iletebilirsiniz.

Bir fonksiyon, sonuç olarak verileri döndürebilir.

3.16.1 Bir Fonksiyon oluşturmak

Python'da bir fonksiyon, `def` anahtar sözcüğünü kullanarak tanımlanır:

Örnek:

```
def my_function():
  print("Hello from a function")
```

3.16.2 Bir Fonksiyon Çağırma

Bir fonksiyonu çağırmak için, fonksiyonun adını ve ardından parantez kullanın:

Örnek:

```
def my_function():
  print("Hello from a function")

my_function()
```

3.16.3 Parametreler

Bilgi fonksiyonlara parametreler yoluyla geçirilebilir.

Parametreler, fonksiyonun adından sonra parantez içinde belirtilir. İstediğiniz kadar parametre ekleyebilirsiniz, sadece virgülle ayırın.

Aşağıdaki örnekte fonksiyonun (`fname`) adında bir parametresi vardır. fonksiyon çağrıldığında, girilen bilgileri fonksiyona parametre olarak göndeririz:

Örnek:

```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

3.16.4 Varsayılan Değerli Parametre

Aşağıdaki örnek, varsayılan bir parametre değerinin nasıl kullanılacağını gösterir. Fonksiyonu parametresiz olarak çağırırsak, varsayılan değeri kullanır:

Örnek:

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

3.16.5 Değer Döndürme

Bir fonksiyonun bir değer döndürmesini sağlamak için `return` ifadesini kullanın:

Örnek:

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

3.16.6 Lambda Fonksiyonları

Python'da, `lambda` anahtar kelimesi anonim fonksiyonlar oluşturmak için kullanılır. Bunlar esas olarak önceden tanımlanmış isimler içermez. Uyarlanabilir fonksiyonlar oluşturmak için iyidir ve bu sayede etkinlik yönetimi için iyidir.

Örnek

i'nin 2 ile çarpım değerini döndüren anonim bir fonksiyon:

```
myfunc = lambda i: i*2
print(myfunc(2))
```

`lambda` tanımlı fonksiyonlar, burada gösterildiği gibi birden fazla tanımlı girişe sahip olabilir:

Örnek:

```
myfunc = lambda x,y: x*y
print(myfunc(3,6))
```

Aşağıdaki örnekte gösterildiği gibi, çalışma zamanında anonim fonksiyonlar oluşturduğunuzda, `lambda`'nın gücü daha iyi anlaşılır.

Örnek:

```
def myfunc(n):
  return lambda i: i*n

doubler = myfunc(2)
tripler = myfunc(3)
val = 11
print("Doubled: " + str(doubler(val)) + ". Tripled: " + str(tripler(val)))
```

Burada `myfunc` adında tanımlanmış fonksiyonu görüyoruz, which creates an anonymous function that doubles some on-the-fly variable i with a just-in-time variable n representing our multiplier.

We then create two variables doubler and tripler, which are assigned to the result of myfunc passing in 2 and 3 respectively. They are assigned to the generated lambda functions.

3.17 Python Dosya Yönetimi

Dosya işleme, herhangi bir web uygulamasının önemli bir parçasıdır. Python, dosyaları oluşturmak, okumak, güncellemek ve silmek için çeşitli fonksiyonlara sahiptir.

3.17.1 Dosya Yönetimi

Python'daki dosyalarla çalışmak için kullanılan anahtar fonksiyon, `open()` fonksiyonudur. `open()` fonksiyonu iki parametre alır; dosya_adı ve modu. Bir dosyayı açmak için dört farklı yöntem (mod) vardır.

- `"r"` - **Read** - Oku - Varsayılan değerdir. Dosya mevcut değilse, okuma için bir dosya açar.
- `"a"` - **Append** - Ekle - Eklemek için bir dosya açar, mevcut değilse dosyayı oluşturur.
- `"w"` - **Write** - Yaz - Yazmak için bir dosya açar, mevcut değilse dosyayı oluşturur.
- `"x"` - **Create** - Oluştur - Belirtilen dosyayı oluşturur, dosya varsa bir hata döndürür.

Ayrıca, dosyanın ikili veya metin modu olarak ele alınıp alınmayacağını belirtebilirsiniz.

- `"t"` - **Text** - Metin - Varsayılan değer. Metin modu
- `"b"` - **Binary** - İkili - İkili mod (ör. resimler)

3.17.2 Söz dizimi

Bir dosyayı okunur olarak açma için dosyanın adını belirtmek yeterlidir:

```
f = open("demofile.txt")
```

Yukarıdaki kod ile aynı işi yapar:

```
f = open("demofile.txt", "rt")
```

Dosya okuma için `"r"` ve dosya modu için `"t"` değerleri varsayılan değerler olduğundan, bunları belirtmeniz gerekmez.

Not: Dosyanın var olduğundan emin olun, aksi takdirde bir hata mesajı alırsınız.

3.18 Python Dosya Açma

3.18.1 Sunucuda bir dosya açma

Python'ı çalıştırdığımız konum ile aynı klasörde bulunan aşağıdaki dosyaya sahip olduğumuzu varsayın:

demofile.txt:

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

Dosyayı açmak için yerleşik `open()` fonksiyonunu kullanın. `open()` fonksiyonu, dosyanın içeriğini okumak için `read()` metoduna sahip bir dosya nesnesi döndürür:

Örnek:

```
f = open("demofile.txt", "r")
print(f.read())
```

3.18.2 Dosyanın belli bir kısmını okuma

Varsayılan olarak `read()` metodu tüm metni döndürür, ancak kaç karakter döndürmek istediğinizi de belirtebilirsiniz:

Örnek

Dosyanın 5 ilk karakterini döndür:

```
f = open("demofile.txt", "r")
print(f.read(5))
```

3.18.3 Satırları Okuma

`readline()` metodunu kullanarak bir satır döndürebilirsiniz:

Örnek

Dosyanın bir satırını okuyun:

```
f = open("demofile.txt", "r")
print(f.readline())
```

`readline()` öğesini iki kez çağırarak, iki ilk satırı okuyabilirsiniz:

Örnek

Dosyanın iki satırını oku:

```
f = open("demofile.txt", "r")
print(f.readline())
print(f.readline())
```

Dosyanın satırları arasında döngü kurarak, tüm dosyayı satır satır okuyabilirsiniz:

Örnek

Dosyayı satır satır okuma:

```
f = open("demofile.txt", "r")
for x in f:
  print(x)
```

3.19 Python Dosya Oluşturma ve Yazma

3.19.1 Varolan bir dosyaya yazma

Var olan bir dosyaya yazmak için open() fonksiyonuna bir parametre eklemelisiniz:

- `"a"` - **Append** - Ekle - dosyanın sonuna eklenir
- `"w"` - **Write** - Yaz - mevcut içeriğin üzerine yazar

Örnek

"Demofile.txt" dosyasını açın ve dosyaya içerik ekleyin:

```
f = open("demofile.txt", "a")
f.write("Now the file has one more line!")
```

Örnek

"Demofile.txt" dosyasını açın ve içeriğin üzerine yazın:

```
f = open("demofile.txt", "w")
f.write("Woops! I have deleted the content!")
```

Not: "w" yöntemi tüm dosyanın üzerine yazacaktır.

3.19.2 Yeni Dosya Oluşturma

Python'da yeni bir dosya oluşturmak için, aşağıdaki parametrelerden biriyle `open()` fonksiyonunu kullanın:

- `"x"` - **Create** - Oluştur - bir dosya oluşturur, dosya mevcutsa bir hata döndürür.
- `"a"` - **Append** - Ek - belirtilen dosya mevcut değilse bir dosya oluşturur.
- `"w"` - **Write** - Yaz - belirtilen dosya mevcut değilse bir dosya oluşturur.

Örnek

"Myfile.txt" adlı bir dosya oluşturun:

```
f = open("myfile.txt", "x")
```

Sonuç: yeni boş bir dosya oluşturuldu!

Örnek

Mevcut değilse yeni bir dosya oluşturun:

```
f = open("myfile.txt", "w")
```

3.20 Python Dosya Silme

3.20.1 Dosya Silme

Bir dosyayı silmek için, OS modülünü içe aktarmanız ve `os.remove()` metodunu çalıştırmanız gerekir:

Örnek

"demofile.txt" dosyasını kaldırın:

```
import os
os.remove("demofile.txt")
```

3.20.2 Dosya var mı kontrol edin:

Bir hata oluşmasını önlemek için, dosyayı silmeye çalışmadan önce mevcut olup olmadığını kontrol etmek isteyebilirsiniz:

Örnek

Dosyanın var olup olmadığını kontrol edin, ardından silin:

```
import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else
  print("Dosya mevcut değil")
```

3.20.3 Klasör Silme

Tüm bir klasörü silmek için `os.rmdir()` metodunu kullanın:

Örnek

"myfolder" klasörünü kaldırın:

```
import os
os.rmdir("myfolder")
```

Not: Bu yöntemle sadece boş klasörleri silebilirsiniz.

3.21 Authors

- Özgür BAYRAKTAROĞLU
- Hürol YALÇIN

3.22 Contributing

- @ozgurturkiye (Özgür Bayraktaroğlu)
- @hurturkiye (Hürol Yalçın)

3.23 History

Here is a history of the project

3.24 Installation

Install the package with pip:

```
$ pip install read-the-docs-template
$ pip install new-try
```

This is editted line

- Bir
- İki

3.25 Python Öğren Belgelerine Hoşgeldiniz!

python-ogren projesi https://www.w3schools.com/python/ adresindeki anlatımların çevirilerini içerir. Python hakkında ek bir Türkçe kaynak olmayı hedefler.

Belgelendirmeye erişmek için: http://python-ogren.readthedocs.io

3.25.1 Dikkat ediniz

- Çeviriler henüz düzenleme aşamasındadır.
- Program kodlarının Türkçeleştirilmesi sırasında hatalar olabilir.
- Çeviriler henüz üçüncü bir göz ile inclenmediği için karmaşık gelebilir.

3.25.2 Katkıda bulunun

- Düzenleme İsteği: https://github.com/ozgurturkiye/python-ogren/pulls
- Kaynak Code: https://github.com/ozgurturkiye/python-ogren
- Belgenin özgün kaynağı: https://www.w3schools.com/python/

3.25.3 Destek

Eğer katkıda bulunmak isterseniz lütfen bize bildirin. İletişim için: ozgurturkiye@gmail.com, hurolyalcin@gmail.com

3.26 Usage

To use this template, simply update it:

```
import read-the-docs-template
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search