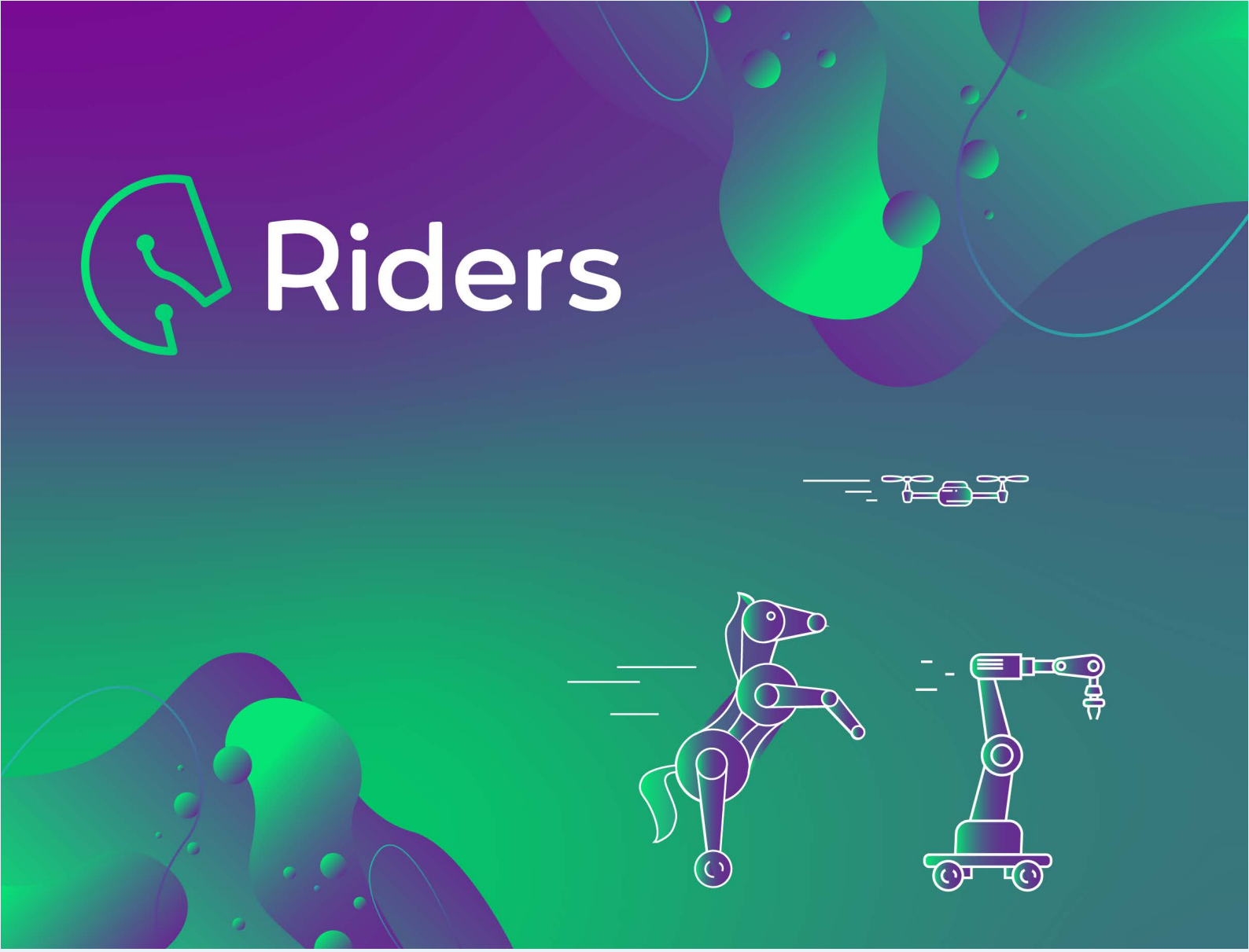

Python
Temel Kavramlar Kılavuzu

Python 3, web geliştiricileri, veri bilimcileri ve yazılım mühendisleri tarafından sevilen, gerçekten çok yönlü bir programlama dilidir. Böyle olmasının birkaç sebebi var!

- Python açık kaynaklıdır ve harika bir destek topluluğuna sahiptir,
- Ayrıca, kapsamlı destek kütüphaneleri vardır,
- Veri yapıları kullanıcı dostudur.

Bir kez alıştığınızda, geliştirme hızınız ve üretkenliğiniz artacak!

İçindekiler

Python Temelleri: Başlangıç

Çoğu Windows ve Mac bilgisayarda Python önceden yüklenmiş olarak gelir. Bunu bir Komut Satırı aramasıyla kontrol edebilirsiniz. Python'un özel cazibesi, herhangi bir metin düzenleyicide bir program yazabilmeniz, .py biçiminde kaydedebilmeniz ve ardından bir Komut Satırı aracılığıyla çalıştırabilmenizdir. Ancak daha karmaşık kodlar yazmayı öğrendikçe veya veri bilimine giriş yaptıkça, bir IDE veya IDLE'ye geçmek isteyebilirsiniz.

IDLE (Entegre Gelişim ve Öğrenme) nedir?

IDLE (Integrated Development and Learning Environment) her Python kurulumuyla birlikte gelir. Diğer metin editörlerine göre avantajı, önemli anahtar kelimeleri (örneğin dize fonksiyonları) vurgulayarak kodu yorumlamanızı kolaylaştırmasıdır.
Shell, Python IDLE için varsayılan çalışma modudur. Özünde, aşağıdaki dört adımı gerçekleştiren basit bir döngüdür:

- Python kodunu okur
- Sonuçlarını değerlendirir
- Sonucu ekrana yazdırır
- Bir sonraki ifadeyi okumak için geri döner.

Python shell, çeşitli küçük kod parçacıklarını test etmek için harika bir yerdir.

Temel Python Veri Türleri

Python'da her değer "nesne" olarak adlandırılır. Ve her nesnenin belirli bir veri tipi vardır. En çok kullanılan üç veri tipi aşağıdaki gibidir:

Tam sayılar (int) - "3 sayısı" gibi bir nesneyi temsil eden bir tam sayıdır.

Tam Sayılar	`-2, -1, 0, 1, 2, 3, 4, 5`

Floating-point sayıları (float) - floating-point sayılarını temsil etmek için kullanılır.

Floating-point Sayıları	`-1.25, -1.0, --0.5, 0.0, 0.5, 1.0, 1.25`

Dizeler — bir karakter dizisini bir dize kullanarak kodlar. Örneğin, "merhaba" kelimesi. Python 3'te dizeler değişmezdir. Eğer bir dize tanımladıysanız, onu daha sonra değiştiremezsiniz.

Bir karakter dizisini replace() veya join() gibi komutlarla değiştirebilirsiniz, ancak bu komutlar karakter dizisinin bir kopyasını oluşturur ve orijinalini yeniden yazmak yerine ona değişiklik uygular.

Dizeler	**‘yo’, ‘hey’, ‘Merhaba!’, ‘Selam!’**

Ayrıca, bahsetmeye değer diğer üç tür de listeler, sözlükler ve tuple'lardır. Bunların hepsi sonraki bölümlerde ele alınacaktır.

Şimdilik dizelere odaklanalım.

Python'da Bir Dize Nasıl Oluşturulur?

Bir dizeyi tek, çift veya üçlü tırnak kullanarak üç şekilde oluşturabilirsiniz. İşte her seçeneğin bir örneği:

Basit Python Dizesi

```
my_string = “Haydi Python Ogrenelim!”
another_string = ‘Basta zor gozukebilir, ama yapabilirsiniz!’
a_long_string = ‘’’Evet, biraz pratik yaparak birden fazla satiri kapsayan cok satirli dizilerde bile ustalasabilirsiniz.’’’
```

DİKKAT! Hangi seçeneği seçerseniz seçin, ona sadık kalmalı ve programınızda tutarlı bir şekilde kullanmalısınız.

Bir sonraki adım olarak, dizenizi konsol penceresinde çıktılamak için print() fonksiyonunu kullanabilirsiniz. Bu, kodunuzu gözden geçirmenize ve her şeyin iyi çalıştığından emin olmanıza olanak tanır.

İşte bunun için bir örnek:

```
print(“Haydi bir dize cikaralim!”)
```

Dize Birleştirme

Ustalaşabileceğiniz bir sonraki şey, "+" operatörünü kullanarak iki dizeyi birbirine eklemenin bir yolu olan birleştirme işlemidir. İşte nasıl yapıldığı:

```
string_one = “Yeni bir kitap “
string_two =  “okuyorum!”
string_three = string_one + string_two
```

Not: Dize + tam sayı gibi iki farklı veri türüne + operatörünü uygulayamazsınız. Bunu yapmaya çalışırsanız, aşağıdaki Python hatasını alırsınız:

```
TypeError: 'İnt' nesnesi dolayli olarak str'ye dönüştürülemiyor
```

Dize Çoğaltma

Adından da anlaşılacağı gibi, bu komut aynı dizeyi birkaç kez tekrarlamanızı sağlar. Bu işlem * operatörü kullanılarak yapılır. Bu operatörün yalnızca dize veri türlerinde çoğaltıcı görevi gördüğünü unutmayın. Sayılara uygulandığında, bir çarpan görevi görür.

Dize çoğaltma örneği:

```
'Alice'  *  5 'AliceAliceAliceAliceAlice'
```

Ve print () ile

```
print ("Alice" * 5)
```

Dolayısıyla çıktınız Alice'in arka arkaya beş kez yazılması olacaktır.

Matematik Operatörleri

Referans olarak, sayılara uygulayabileceğiniz diğer matematik işlemlerinin bir listesini burada bulabilirsiniz:

Operatörler	İşlemler	Örnek
**	Üstel	2 ** 3 = 8
%	Modül/Örgü	22 % 8 = 6
//	Tam sayı bölme	22 // 8 = 2
/	Bölme	22 / 8 = 2.75
*	Çarpma	3 * 3 = 9
-	Çıkarma	5 – 2 = 3
+	Toplama	2 + 2 = 4

Dizeler Değişkenlerde Nasıl Saklanır?

Python 3'teki değişkenler, kendisine bağlı olan bir değere belirli bir depolama konumu atayan özel sembollerdir. Özünde, değişkenler bir değerin nerede saklandığını bilmek için üzerine yerleştirdiğiniz özel etiketler gibidir.

Dizeler veri içerir. Böylece onları bir değişken içinde "paketleyebilirsiniz". Bunu yaparak kodunuzun okunabilirliğini arttırabilirsiniz.

Bir dizeyi bir değişken içinde nasıl saklayabileceğiniz aşağıda açıklanmıştır.

```
my_str = “Herkese merhaba”
```

Bunu biraz daha açalım:

- my_str değişken adıdır.
- = atama operatörü.
- String değişkene atadığımız bir değerdir

Bunu yazdırdığınızda, dize çıktısını alırsınız.

```
print(my_str)
```

```
= Herkese merhaba
```

Gördünüz mü? Değişkenleri kullanarak, her kullanmak istediğinizde tüm dizeyi yeniden yazmanız gerekmediğinden, kendinizi yığınla çabadan kurtarırsınız.

Python'da Built-in Fonksiyonları

Python'daki en popüler fonksiyonu zaten biliyorsunuz - print(). Şimdi platformda yerleşik olarak bulunan eşit derecede popüler kuzenlerine bir göz atalım.

Input() Fonksiyonu

input() fonksiyonu kullanıcıdan bazı girdiler istemenin basit bir yoludur (örneğin, isim vermek). Tüm kullanıcı girdileri bir dize olarak saklanır.

İşte bunu göstermek için kısa bir örnek:

```
name = input(“Selam! Adin ne? “)
print(“Tanistigima memnun oldum “ + name + “!”)

age = input(“Kaç yasindasin “)
print(“Yani, simdiden“ + str(age) + “ yasindasin,
“ + name + “!”)
```

Bu kısa programı çalıştırdığınızda, sonuçlar aşağıdaki gibi görünecektir:

Selam! Adın ne? “Arda”

Tanıştığıma memnun oldum, Arda!

Kaç yaşındasın?? 25

Yani, şimdiden 25 yaşındasın, Arda!

len() Fonksiyonu

len() fonksiyonu, herhangi bir dize, liste, tuple, sözlük veya başka bir veri türünün uzunluğunu bulmanıza yardımcı olur. Aşırı değerleri belirlemek ve programınızın performansını optimize etmek için bunları kırpmak için kullanışlı bir komuttur.

İşte bir dize için bir giriş fonksiyonu örneği:

```
# len() testi
str1 =  “Umarim egitimimizi begenmissinizdir!”
print(“Dizenin uzunlugu:”, len(str1))
```

Çıktı:

Dizenin uzunluğu: 35

filter() - Filtreleme

Yinelenebilir bir nesnede (listeler, tuple'lar, sözlükler, vb.) bazı öğeleri hariç tutmak için **filter()** fonksiyonunu kullanın

```
ages =  5, 12, 17, 18, 24, 32

def myFunc (x):
  if  x < 18:
      return False
  else:
      return True

  adults = filter(myFunc, ages)

  for x in adults:
      print(x)
```

(İsteğe bağlı: Kontrol listesinin PDF versiyonu, tüm dahili işlevlerin tam bir tablosunu da içerebilir).

Bir Fonksiyon Nasıl Tanımlanır?

Python 3, dahili fonksiyonları kullanmanın yanı sıra, programınız için kendi fonksiyonlarınızı tanımlamanıza da izin verir.
Özetlemek gerekirse, bir **fonksiyon** belirli bir eylemi gerçekleştiren kodlanmış talimatlar bloğudur. Düzgün bir şekilde tanımlandıktan sonra, bir fonksiyon programınız boyunca tekrar kullanılabilir, yani aynı kodu tekrar kullanabilirsiniz.

İşte Python'da bir fonksiyonun nasıl tanımlanacağını açıklayan kısa bir kılavuz:

İlk olarak **def** anahtar kelimesini ve ardından **name()** fonksiyonunu kullanın: Parantezler, fonksiyonunuzun alması gereken parametreleri içerebilir.

```
def name():
```

Ardından, bu fonksiyonun ne yapması gerektiğini belirtmek için 4 aralıklı bir girintiyle ikinci bir kod satırı eklemeniz gerekir.

```
def name():
    print("Adin ne?")
```

Şimdi, kodu çalıştırmak için bu fonksiyonu çağırmanız gerekir.

```
name.py
def name():
    print("Adin ne?")

name()
```

Şimdi, bir parametre ile tanımlanmış bir fonksiyona bir göz atalım - bir fonksiyonun kabul edebileceği argümanı belirten bir özellik.

```
def add_numbers(x, y, z):
   a = x + y
   b = x + z
   c = y + z
  print(a, b, c)

 add_numbers(1, 2, 3)
```

Bu durumda, x parametresi için 1, y parametresi için 2 ve z parametresi için 3 sayısını girersiniz. Program sayıları toplamak gibi basit bir matematik işlemi yapacaktır:

Çıktı:

```
a = 3
b = 4
c = 5
```

Anahtar Kelime Argümanları Bir Fonksiyona Aktarma

Bir fonksiyon anahtar kelime argümanlarını da kabul edebilir. Bu durumda, Python yorumlayıcısı değerleri parametrelerle eşleştirmek için sağlanan anahtar kelimeleri kullanacağından, parametreleri rastgele sırayla kullanabilirsiniz.

İşte bir anahtar kelime argümanını bir işleve nasıl aktaracağınıza dair basit bir örnek:

```
# Fonksiyonu parametrelerle tanımlayın
def product_info (ürün adı, fiyat):
    print(“Urun Adi: “ + product_name)
    print(”Fiyat: “ + str(price))
    # Fonksiyonu aşağıdaki gibi üretin:
    product_info("Beyaz Tisort: ", 15)

    # Anahtar kelime değişkenleriyle fonksiyon üretin
    product_info(productname=”Pantolon”, price=45)
```

Çıktı:

```
Ürün adı: Beyaz Tişört
Fiyat:15
Ürün adı: Pantolon
Fiyat:45
```

Listeler

Listeler, Python'da sıralı bir liste belirtmek için kullanılan başka bir mihenk taşı veri türüdür. Kısacası, ilgili verileri bir arada tutmanıza ve aynı anda birkaç değer üzerinde gerçekleştirmenize yardımcı olurlar. Dizelerin aksine listeler değiştirilebilir (=changeable).

Bir liste içindeki her bir değere **öğe** denir ve bunlar köşeli parantezler arasına yerleştirilir.

Liste Örnekleri

```
my_list =  [1, 2, 3]
my_list =  [“a”, “b”, “c”]
my_list =  [“4”, d, “kitap”, 5]
```

Alternatif olarak, aynı işlemi yapmak için **list()** işlevini kullanabilirsiniz:

```
alpha_list = list ((“1”, “2”, “3”))
print(alpha_list)
```

Bir Listeye Nasıl Öğe Eklenir?

Mevcut listelere yeni öğeler eklemek için iki yolunuz vardır.

Bunlardan ilki **append()** fonksiyonunu kullanmaktır:

```
beta_list = [“elma”, “muz”, “portakal”]
beta_list.append(“uzum”)
print(beta_list)
```

İkinci seçenek, belirtilen indekse bir öğe eklemek için **insert()** f onksiyonunu kullanmaktır:

```
beta_list =  [“elma”, “muz”, “portakal”]
beta_list.insert(2,“uzum”)
print(beta_list)
```

Listeden Bir Öğe Nasıl Kaldırılır?

Bunu yapmanın birkaç yolu vardır. İlk olarak **remove()** fonksiyonu kullanılabilir:

```
beta_list = [“elma”, “muz”, “portakal”]
beta_list.remove(“elma”)
print(beta_list)
```

İkinci olarak, **pop()** fonksiyonunu kullanabilirsiniz. Herhangi bir dizin belirtilmemişse, son öğeyi kaldıracaktır.

```
beta_list = [“elma”, “muz”, “portakal”]
beta_list.pop()
print(beta_list)
```

Son seçenek, belirli bir öğeyi kaldırmak için **del anahtar kelimesini** kullanmaktır:

```
beta_list = [“elma”, “muz”, “portakal”]
del beta_list  [1]
print(beta_list)
```

Not: Listeyi tamamen silmek için tüm listeye del komuıtunu uygulayabilirsiniz.

İki Listeyi Birleştirme

İki listeyi karıştırmak için + komutunu kullanın.

```
my_list = [1, 2, 3]
my_list =  [“a”, “b”, “c”]
combo_list = my_list + my_list2
combo_list
[1, 2, 3, ‘a’, ‘b’, ‘c’]
```

İç İçe Liste Oluşturma

Bol miktarda listeniz olduğunda listelerinizin bir listesini de oluşturabilirsiniz :)

```
my_nested_list = [my_list, my_list2]
my_nested_list
[[1, 2, 3], [‘a’, ‘b’, ‘c’]]
```

Liste Sıralama

Listenizdeki tüm öğeleri düzenlemek için **sort()** f onksiyonunu kullanın.

```
alpha_list =  [34, 23, 67, 100, 88, 2]
alpha_list.sort()
alpha_list
[2, 23, 34, 67, 88, 100]
```

Liste Ayırma

Şimdi, listenizden sadece birkaç öğeyi çağırmak istiyorsanız (örneğin ilk 4 öğe), iki nokta üst üste [x:y] ile ayrılmış bir dizi indis numarası belirtmeniz gerekir. İşte bir örnek:

```
alpha_list [0:4]
[2, 23, 34, 67]
```

Listedeki Öğelerin Değerini Değiştirin

Bir liste öğesinin değerinin üzerine kolayca yazabilirsiniz:

```
beta_list = [“elma”, “muz”, “portakal”]
beta_list 1 =  “armut”
print(beta_list)
```

Çıktı:

```
[‘elma’, ‘armut’, ‘kiraz’]
```

Liste İçinde Döngü

for döngüsünü kullanarak, * operatörünün yaptığına benzer şekilde, belirli öğelerin kullanımını çoğaltabilirsiniz. İşte bir örnek:

```
for x in range(1,4):
beta_list +=  [‘meyve’]
print(beta_list)
```

Liste Kopyalama

Verilerinizi çoğaltmak için yerleşik **copy**() işlevini kullanın:

```
beta_list = [“elma”, “muz”, “portakal”]
beta_list = beta_list.copy()
print(beta_list)
```

Alternatif olarak, **list**() yöntemiyle bir listeyi kopyalayabilirsiniz:

```
beta_list = [“elma”, “muz”, “portakal”]
beta_list = list (beta_list)
print(beta_list)
```

Liste Kavramaları

Liste kavramaları, mevcut listelere dayalı listeler oluşturmak için kullanışlı bir seçenektir. Bunları kullanırken dizeler ve tuple'lar kullanarak da oluşturabilirsiniz.

Liste kavrama örnekleri

```
list_variable = [x for x in iterable]
```

Matematik işlemleri, tam sayılar ve **range**() fonksiyonunu içeren daha karmaşık bir örnek:

```
number_list = [x** 2 for x in range(10) if x % 2 == 0]
print(number_list)
```

Tuple'lar

Tuple'lar listelere benzer - öğelerin sıralı bir dizisini görüntülemenizi sağlarlar. Ancak, değişmezdirler ve bir tuple'da depolanan değerleri değiştiremezsiniz.

Listeler yerine tuple kullanmanın avantajı, ilkinin biraz daha hızlı olmasıdır. Bu yüzden kodunuzu optimize etmenin güzel bir yoludur.

Bir Tuple Nasıl Oluşturulur?

```
my_tuple = (1, 2, 3, 4, 5)
my_tuple[0:3]
(1, 2, 3)
```

Not: Bir tuple oluşturunca, ona yeni öğeler ekleyemez veya değiştiremezsiniz!

Bir Tuple Nasıl Kaydırılır?

Süreç, listeleri ayırmaya benzer.

```
numbers = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
print(numbers[1:11:2])
```

Çıktı:

```
(1, 3, 5, 7, 9)
```

Tuple'ı Listeye Dönüştürme

Tuple'lar değişmez olduğu için onları değiştiremezsiniz. Bir tuple'ı bir listeye dönüştürebilir, düzenleyebilir ve ardından tekrar bir tuple'a dönüştürebilirsiniz.

Bunu nasıl yapabileceğinizi anlatalım:

```
x =( “elma”, “portakal”, “armut”)
y = list(x)
y[1]= “uzum”
x = tuple(y)
print(x)
```

Sözlükler

Bir sözlük, belirli değerlerle eşlenen anahtarlara sahip dizinleri tutar. Bu anahtar-değer çiftleri Python'da veri düzenlemek ve depolamak için harika bir yol sunar. Değiştirilebilirler, yani depolanan bilgileri değiştirebilirsiniz.

Anahtar değeri bir **dize**, **Boolean** veya **tam sayı** olabilir. İşte bunu gösteren örnek bir sözlük:

```
Customer 1= {‘kullanici adi’: ‘john-sea’, ‘online’:
false, ‘arkadaslar’:100}
```

Python Sözlüğü Nasıl Oluşturulur?

Boş bir sözlüğün nasıl oluşturulacağını gösteren kısa bir örnek:

Seçenek 1: `new_dict = {}`

Seçenek 2: `other_dict= dict()`

Sözlüğünüze değer eklemek için de aynı iki yaklaşımı kullanabilirsiniz:

```
new_dict = {
 “marka”: “Honda”,
 “model”: “Civic”
 ,“yil”:  1995
}
print(new_dict)
```

Sözlükte Bir Değere Nasıl Erişilir?

Sözlüğünüzdeki herhangi bir değere aşağıdaki şekilde erişebilirsiniz:

```
= new_dict[“marka”]
```

Aynı işlemi gerçekleştirmek için aşağıdaki yöntemleri de kullanabilirsiniz.

- **`dict.keys()`** anahtarları izole eder
- **`dict.values()`** değerleri izole eder
- **`dict.items()`** öğeleri (anahtar, değer) tuple çiftlerinden oluşan bir liste biçiminde döndürür

Öğe Değerini Değiştirme

Öğelerden birini değiştirmek için ona anahtar adıyla başvurmanız gerekir:

```
#"Yılı" 2020'ye değiştir:

new_dict= {
 “marka”: “Honda”,
 “model”:  “Civic”,
 “yil”: 1995
}
new_dict[“yil”] =  2020
```

Sözlük İçinde Döngü Komutu

Döngü uygulamak için **for** döngü komutunu kullanın.

Not: Bu durumda, geri dönüş değerleri sözlüğün anahtarlarıdır. Ancak, başka bir yöntem kullanarak da değerleri döndürebilirsiniz.

```
#sözlükteki tüm anahtar adlarını yazdır

for x in new_dict:
 print(x)

#sözlükteki tüm değerleri yazdır

for x in new_dict:
 print[new_dict(x)]

#hem anahtarlar hem de değerler arasında döngü

for x, y in my_dict.items():
 print(x, y)
```

Python'da If İfadeleri (Koşullu İfadeler)

Tıpkı diğer programlama dilleri gibi Python da matematikteki temel mantıksal koşulları destekler:

- Eşitlik: a == b
- Eşit değil: a != b
- Küçüktür: a < b
- Küçük eşittir: a <= b
- Büyüktür: a > b
- Büyük eşittir: a >= b

Bu koşullardan çeşitli şekillerde yararlanabilirsiniz. Ancak büyük olasılıkla bunları "**if ifadelerinde**" ve **döngülerde** kullanacaksınız.

If İfadesi Örneği

Bir koşullu ifadenin amacı, doğru veya yanlış olup olmadığını kontrol etmektir.

```
if 5 > 1:
      print(“Dogru!”)
```

Çıktı:

```
Doğru!
```

İç İçe If İfadeleri

Daha karmaşık işlemler için iç içe if deyimleri oluşturabilirsiniz. İşte böyle görünüyor:

```
x = 35

if x > 20:
   print(“Yirmiden buyuk,”)
   if x > 30:
      print(“ve 30dan da buyuk!”)
```

Elif İfadeleri

elif anahtar sözcüğü, bir önceki koşul(lar) doğru değilse programınızın başka bir koşulu denemesini ister. İşte bir örnek:

```
a= 45
b= 45
if b > a:
  print(“b a'dan buyuk”)
elif a == b:
  print(“a ve b esit”)
```

If Else İfadeleri

else anahtar kelimesi, koşul cümlenize bazı ek filtreler eklemenize yardımcı olur. Bir if-elif-else kombinasyonu şu şekilde görünür:

```
if age < 4:
     ticket_price = 0
elif age < 18:
     ticket_price = 10
else:
     ticket_price = 15
```

If-Not İfadeleri

Not anahtar sözcüğü, değerin doğru olmadığını doğrulamak için zıt anlamı kontrol etmenizi sağlar:

```
new_list = [1, 2, 3, 4]
x = 10
if x not in new_list:
  print(“’x’ listede degil, yani dogru!”)
```

Pass İfadeleri

If ifadeleri boş olamaz. Ancak öyle oluyorsa, hata almaktan kaçınmak için **pass** ifadesini ekleyin:

```
a = 33
b = 200

if b > a:
 pass
```

Python Döngüleri

Python'da bilinmesi gereken iki basit döngü komutu vardır:

- for döngüleri
- while döngüleri

Şimdi bunların her birine bir göz atalım.

For Döngüsü

Bu Python kontrol listesinin diğer bölümlerinde daha önce gösterildiği gibi, for döngüsü liste, tuple, sözlük, dize vb. gibi bir dizi üzerinde yineleme yapmak için kullanışlı bir yoldur. Bir dize boyunca nasıl döngü yapılacağını gösteren bir örnek:

```
for x in “elma”:
 print(x)
```

Ayrıca, listeler ve sözlükler için başka örnekler de gördünüz.

While Döngüsü

While döngüsü, koşul doğru olduğu sürece bir dizi deyimi yürütmenizi sağlar.

```
#x 8'den küçük olduğu sürece yazdır

i = 1
while  i < 8:
    print(x)
    i +=  1
```

Bir Döngü Nasıl Bozulur?

Ayrıca koşul karşılansa bile döngünün çalışmasını durdurabilirsiniz. Bunun için hem while hem de for döngülerinde break deyimini kullanın:

```
i = 1
while  i < 8:
   print(i)
   if i == 4:
     break
   i +=  1
```

Sınıflar

Python nesne yönelimli bir programlama dili olduğu için yöntemleri ve özellikleriyle birlikte neredeyse her öğesi bir **nesnedir**.

Sınıflar, farklı nesneler oluşturmak için bir plan görevi görür. **Nesneler**, sınıfın bir programda tezahür ettiği bir sınıf örneğidir.

Nasıl Sınıf Oluşturulur?

TestClass adında, z özelliğe sahip bir sınıf oluşturalım:

```
class Testclass:
    x =  5
```

Nasıl Nesne Oluşturulur?

Bir sonraki adım olarak, sınıfınızı kullanarak bir nesne oluşturabilirsiniz. Örneğin:

```
p1 = Testclass()
print(p1.x)
```

Ayrıca, nesnenize farklı nitelikler ve yöntemler atayabilirsiniz. Aşağıdaki örnek gibi:

```
class car(object):
    """docstring"""

    def __init__(self, color, doors, tires):
        """İnsaatci"""
       self.color = color
       self.doors = doors
       self.tires = tires

     def brake(self):
         """
         Arabayi durdur
         """
         return "Frenleme"

   def drive(self):
        """
        Arabayi sur
        """
        return "Suruyorum!"
```

Alt Sınıf Nasıl Oluşturulur?

Her nesne daha fazla alt sınıflandırmaya tabi tutulabilir. Örnek olarak:

```
class Car(Vehicle):
    """
    Araba sinifi
    """
    def brake(self):
        """
        Fren yontemini gecersiz kil
        """
        return "Araba yavasca frenliyor!"
if __name__ == "__main__":
    car = Car("sarı", 2, 4, "araba")
    car.brake()
    'Araba yavasca frenliyor!'
    car.drive()
    "Sari bir araba suruyorum!"
```

Python Hataları ile Başa Çıkma

Python, kodunuzda bir hata yaptığınızda ortaya çıkacak yerleşik bir istisnalar (hatalar) listesine sahiptir. Yeni biri için, bunları nasıl düzelteceğinizi bilmek iyidir.

En Yaygın Python İstisnaları

- `AttributeError` — bir nitelik referansı veya ataması başarısız olduğunda açılır.
- `IOError` — bir I/O işlemi (örneğin bir open() işlevi) I/O ile ilgili bir nedenden dolayı başarısız olduğunda ortaya çıkar, örneğin "dosya bulunamadı" veya "disk dolu".
- `ImportError` — bir import ifadesi modül tanımını bulamadığında ortaya çıkar. Ayrıca, bir "from..." import ifadesi içe aktarılması gereken bir ismi bulamadığında.
- `IndexError` — bir dizi alt simgesi aralık dışında olduğunda ortaya çıkar.
- `KeyError` — bir sözlük anahtarı mevcut anahtarlar kümesinde bulunamadığında yükseltilir.
- `KeyboardInterrupt` — kullanıcı kesme tuşuna bastığında (Control-C veya Delete gibi) çıkar.
- `NameError` — yerel veya genel bir ad bulunamadığında görüntülenir.

- `OSError` — sistemle ilgili bir hata olduğunu gösterir.
- `SyntaxError` — ayrıştırıcı bir sözdizimi hatasıyla karşılaştığında açılır.
- `TypeError` — uygun olmayan türdeki bir nesneye bir işlem veya işlev uygulandığında ortaya çıkar.
- `ValueError` — Yerleşik bir işlem/fonksiyon doğru türe sahip ancak uygun değere sahip olmayan bir argüman aldığında ve durum IndexError gibi daha kesin bir istisna tarafından tanımlanmadığında yükseltilir.
- `ZeroDivisionError` — bir bölme veya modül işleminin ikinci argümanı sıfır olduğunda ortaya çıkar.

Hatalar Nasıl Giderilir?

Python, sadece istisnaları ele almak amacıyla tasarlanmış kullanışlı bir -try/except deyimine sahiptir. Bu deyimi kullanarak bir sözlükteki KeyError'leri nasıl yakalayabileceğinizi gösteren örneğe bakalım:

```
my_dict = {“a”:1, “b”:2, “c”:3}
try:
    value = my_dict[“d”]
except  KeyError:
    print(“Boyle bir anahtar yok!”)
```

Ayrıca tek bir deyimle birkaç istisnayı aynı anda tespit edebilirsiniz.
Örnek olarak:

```
my_dict = {“a”:1, “b”:2, “c”:3}
try:
    value = my_dict[“d”]
except  IndexError:
    print(“Boyle bir endeks yok!”
except  KeyError:
    print(“Bu anahtar sozlukte yok!”)
except:
   print(“Baska bir problem ortaya cikti!”)
```

try/except ile else ifadesi

Bir else ifadesi eklemek, hiçbir hata bulunmadığını doğrulamanıza yardımcı olacaktır:

```
my_dict = {“a”:1, “b”:2, “c”:3}

try:
    value = my_dict[“a”]
except  KeyError:
    print(“Bir KeyError ortaya cikti!”)
else:
    print(“Bir hata bulunmadi!”)
```

Sonuçlar

Artık temel Python kavramlarını biliyorsunuz!

Bu Python temel kavramlar kılavuzu çok kapsamlı değildir. Ancak yeni başlayan biri olarak öğrenmeniz gereken tüm temel veri türlerini, işlevleri ve komutları içerir.

Her zaman olduğu gibi, aşağıdaki yorum bölümünde görüşlerinizi bekliyoruz!


Riders