Al Ruby'i Vur Python'a

Python Tutorial'ı Üzerinde Gezintiler

Erek Göktürk – Onur Küçük

Özgür Yazılım A.Ş.
www.ozguryazilim.com.tr

Çalıştırma

- Etkileşimli: irb
- Argümanlar: ARGV[n]

- Etkileşimli: python
- Argümanlar: sys.argv[n]

Karmaşık Sayılar

- Özel bir sözdizimi yok. Complex sınıfı var:
  Complex('4+3i')
- Sözdizimi:
  4+3j
  Hatta:
  1+1j*3 = 3+3j !!!

Dizgeler

- Raw string r”” yok.
- “”” ve ''' yok. Bash tipi heredoc var.
- Substring almak range nesneleri kullanarak:
  “abcd”[1..2]
  (1..2).to_a #=> [1,2]
- Substring almak split notasyonuyla (syntactic):
  “abcd”[1:2]

Diziler

- Range ile indexleme
- Split özel sözdizimi

Sözdizimi

- Anahtar kelimeler blokları sınırlıyor:

```
if a > b
  sunu_bunu
elsif b > c
  bunlar_sunlar
else
  bunu_sunu
end
```

- Anahtar kelimeler ve indentation blokları sınırlıyor:

```
if a > b:
    sunu_bunu
elif b > c:
    bunlar_sunlar
else:
    bunu_sunu
```

İterasyon

- [1,2,3].each do |x|
  f(x)
  end
- (1..3)
- (1..3).to_a
- (1..10).step(3) do |x|
  f x
  end

- for x in [1,2,3]:
  f(x)
- range(1,3)
- list(range(1,3))
- for x in range(1,10,3):
  f(x)

İterasyon - 2

- Döngülerde “else” yok. break ve continue çok kullanılmıyor.

- Pass yok. Boş bıraksak?
  class A
  end

- Döngülerde else var:

```
for x in [1,2,3]:
    if x < 0:
        print('neg!')
        break
else:
    print('neg yok!')
```

- pass var:
  class A:
  pass

Fonksiyonlar

- Rubydoc
  ```
  # işbu fonksiyon ...
  def bufonk
    ...
  end
  ```
- Global'lara değer atanabiliyor.
- Call by reference
- Return değer dönmek için zorunlu değil.

- Docstring
  ```
  def bufonk
      “işbu fonksiyon”
      ...
  ```
- Global'lara doğrudan değer atanamıyor (?)
- Call by reference
- Return değer dönmek için zorunlu.

Fonksiyonlar - 2

- Fonksiyon diye bir veri tipi yok.

```
def a
  “a”
end
b = a
b.class #=> String
b() #=> error!
```

- Yeniden isimlendirme:

```
alias_method a b
```

- Fonksiyon diye veri tipi var.

```
def a():
    return “a”
b = a
b()  #=> “a”
```

Fonksiyonlar - 3

- Parametre varsayılan değeri fonksiyon her çağrıldığında yeniden hesaplanıyor:

```
b = “zonk”
def a(p = b)
  p
end
a   #=> “zonk”
b = “zork”
a   #=> “zork”
```

- Parametre varsayılan değeri sadece fonksiyonun tanımlandığı noktada hesaplanıyor:

```
b = “zonk”
def a(p = b):
    return p
a()  #=> “zonk”
b = “zork”
a()  #=> “zonk”
```

Fonksiyonlar - 4

- İsimlendirilmiş parametreler yok, yedirilmiş sözlükler var.

```
def a(b, nmd)
  .....
end
a("X")
a(:c => "XX") #=>hata
a("X", :c => "XX")
a("X", {:c => "XX"})
```

- İsimlendirilmiş parametreler var.

```
def a(b, c="zip"):
    .....

a("X")
a(c="XX")
a("X", "XX")
```

Fonksiyonlar - 5

- Değişken sayıda argüman alan fonksiyonlar aynı.
  def a(b, *args)
    ....
  end
- Python'da şu yok (?):
  def a(b, c, d) .... end
  b = [1, 2, 3]
  a(*b)

- Değişken sayıda argüman alan fonksiyonlar aynı.
  def a(b, *args)
    .....

Lambda Fonksiyonlar

- Lambda fonksiyon'lar closure.

- Python tutorial'dan:
  By popular demand, a few features commonly found in functional programming languages like Lisp have been added to Python. ...
  Semantically, they are just syntactic sugar for a normal function definition. Like nested function definitions, lambda forms can reference variables from the containing scope.

- Lambda fonksiyonlar closure.

Lambda Fonksiyonlar - 2

- ```
  def z(lmb)
    lmb.call(7)
  end
  y = 3
  a = lambda { |x| x+y }
  a.call(5) #=> 8
  y = 5
  a[5] #=> 10
  z(a) #=> 12
  ```

- ```
  def z(lmb):
      return lmb(7)

  y = 3
  a = lambda x: x+y
  a(5)  #=> 8
  y = 5
  a(5)  #=> 10
  z(a) #=> 12
  ```

Kod Yazma Stili

- Standart yok
- Indent: 2 boşluk (soft)
- Satırda maks 80 veya 120 karakter (öneririm)
- CamelCase sınıflar
- Büyük harfle başlayan her şey sabit, ama sabitler tamamen büyük harf
- Alt çizgili değişken ve fonksiyon isimleri

- PEP 8
- Indent: 4 boşluk (soft)
- Satırda maks 79
- CamelCase sınıflar
- Alt çizgili değişken ve fonksiyon isimleri

Listeler

- Sözdizimi şekeri:
  %w{'hay', 'huy'}
  #=> ['hay', 'huy']
- Array < Enumeration
  sınıfının metodları

Liste Anlama :)

- a = [1,2,3]
  a.map {|x| x+1}
  #=> [2,3,4]
- a = [1,2,3]
  b = [1,2]
  a.product(b).
  map {|x,y| x+y}

- a = [1,2,3]
  [x+1 for i in a]
  #=> [2,3,4]
- a = [1,2,3]
  b = [1,2]
  [x+y for i in a for j in b]
  #=> [2, 3, 3, 4, 4, 5]