Python Programcıları için JavaScript

Python Saati Buluşması
Pylstanbul

Video Kaydı: https://www.youtube.com/watch?v=EEXFcdTmgZM

Üstün Özgür / 19 Mayıs 2020

19 Mayıs Kutlu Olsun

Saygıyla anıyoruz

JavaScript
1995

- JavaScript ya da JS
- ECMAScript resmi standardı
  - Dildeki temel özellikler
  - ES6: ES2015
  - ES2019, ES2020...
- Native ve host
- Dilin kendisinde I/O, networking vs özellikleri yok
- Host tarafından sağlanır.

- JavaScript'in çalıştığı ortamlar
  - Web'in resmi dili
  - node.js
  - Adobe Acrobat
  - React Native
  - GNOME
  - Microsoft Office...

"Brendan Eich Python hakkında ICFP'05 keynote konuşmasının soru-yanıt kısmında olumlu şeyler söylemişti. Hatta büyük Web scriptleri için. Python'ın daha iyi bir tercih olabileceğini yorumunda bulundu. Sonraki yılda ES242'de Python özelliklerini temel alan bazı özelliklerin standarda girmesi için uğraş verdi. Bunlar iteratörler, jeneratörler, destructuring ve array comprehensionlarıydı."

Kaynak: Wirfs-Brock & Eich, "JavaScript: The First 20 Years" https://zenodo.org/record/3707008

Python ile Karşılaştırma

Benzer Noktalar

- Dinamik türlü (dynamically typed)
- Nesne yönelimli
- Birinci sınıf fonksiyonlar
- Karışık paradigmalı (multi-paradigm)
- REPL
- Geliştirme hızı yüksek ve öğrenmesi kolay
- İkisi de çok yaygın ve geleceği parlak diller
- Diğer benzer diller: Ruby, Lua, Lisp ailesi (Clojure, Common Lisp)

Python ile Karşılaştırma
Farklı Noktalar

- Sözdizim olarak C ailesine yakın, girinti yerine süslü parantezler
- Birden fazla resmi geliştirici grubu.
- TC39 (Teknik Komite 39) EcmaScript resmi standardı
- Birden fazla ortam hedefi
- Web sayfalarında tarafından çalışma
- Başlangıçtaki hedef amatör kullanıcıların HTML içinde kullanması

Python ile Karşılaştırma
Farklı Noktalar

- Daha çok motor: V8, JavaScriptCore, Hermes,
- Daha hızlı (çoğu motorda JIT var, Python'da PyPy)
- Not: Tarayıcılardaki JavaScript motoru ve render motoru
- Tarihsel olarak single threaded, günümüzde worker threadleri mümkün
- Genelde asenkron
- Standart kütüphanesi daha zayıf; ama JavaScript + Web API + node.js ile zengin
- Daha kısa bir zamanda geliştirildiği için tehlikeli noktaları daha fazla

JavaScript: Diğer dillerden bir seçki

"Çoğu kütüphane özellikleri diğer yaygın dillerdeki özelliklerden ilham almıştı. array concat ve slice metodları Python dizi operasyonlarından ilham almıştı. Array push, pop... doğrudan Perl'ün array metodlarıydı. Python aynı zamanda String'deki concat, slice, search gibi metodlara ilham verdi. Java, charCodeAt2e ilham verdi. Düzenli ifadelerin sözdizimi ve semantiği ise Perl'den alınmıştı."

Kaynak: Wirfs-Brock & Eich, "JavaScript: The First 20 Years"

JavaScript Nerede Çalışır?

- Bir web sayfasında
  - script etiketi içinde
  - script etiketiyle hedeflenen başka bir dosya
  - Ardışık olarak çalıştırılır
  - Dev Tools - Console ve Snippets
- Sunucu tarafında
  - node

Değişken Tanımlama

JS

- a = 2
  - Global değişken
- var a = 2
  - Fonksiyon düzeyinde tanımlı
  - Fonksiyon içinde bu satıra gelmeden de erişilebilir
- let a = 2
- const a = 2
  - Blok düzeyinde
- *Günümüzde pratikte sadece let ve const kullanılmakta*

Python

- a = 2
  - Modül ya da fonksiyon düzeyinde
- nonlocal a
  - Üstteki bloktaki (ör: fonksiyondaki) değişken
- global a:
  - Modül düzeyindeki değişken

Değişken Tanımlama

JS

- a = 2
  - Global değişken
- var a = 2
  - Fonksiyon düzeyinde tanımlı
  - Fonksiyon içinde bu satıra gelmeden de erişilebilir
- let a = 2
- const a = 2
  - Blok düzeyinde
- *Günümüzde pratikte sadece let ve const kullanılmakta*

Python

- a = 2
  - Modül ya da fonksiyon düzeyinde
- nonlocal a
  - Üstteki bloktaki (ör: fonksiyondaki) değişken
- global a:
  - Modül düzeyindeki değişken

Global Değişkenler

- Tarayıcıda window ya da worker global
- node.js'te global
- Yeni standartta globalThis olacak
- var kullanmayı unutursak global üzerinde tanımlanır. Tehlikeli.
  - *Strict olmayan eski modda ya da modül dışında*
- a = 2
- console.log(window.a)
- Python'da globals() => modül (dosya) düzeyinde

```
a = 1

def foo():

    def bar():
        a = 3
         print(a)

    a = 2
    print(a)
    bar()
    print(a)

print(a)
foo()
print(a)
```

```
a = 1

def foo():
    global a

    def bar():
        global a
        a = 3
        print(a)

    a = 2
    print(a)
    bar()
    print(a)

print(a)
foo()
print(a)
```

```
a = 1

def foo():

    def bar():
        nonlocal a
        a = 3
        print(a)

    a = 2
    print(a)
    bar()
    print(a)

print(a)
foo()
print(a)
```

```
let i = 0

function foo() {
    i = 10
    console.log(i)

    function bar() {
        i = 20
        console.log(i)
    }

    bar()

    console.log(i)

}

console.log(i)

foo()

console.log(i)
```

```
let i = 0;

function foo() {
    let i = 10
    console.log(i)

    function bar() {
       let i = 20
       console.log(i)
    }

    bar()

    console.log(i)

}

console.log(i)

foo()

console.log(i)
```

Türler

JS

- İlkel (Primitive) Türler
  - null
  - undefined
  - Boolean
  - Number IEEE-754 double
  - BigInt
  - Symbol
  - String
- Nesne Türü
  - Object

Python

- Her şey bir nesne (object)

- Bazı nesneler immutable, bazıları mutable
- None
- numerics: boolean, int, double
- sequences: string, list, tuple...
- mappings: dict
- classes
- Instances

Python: Eşitlik ve Tür Dönüşümleri

==: değer eşitliği

- 1 == 1
- [ ] == [ ]
- { } == { }
- [1, 2] == [1, 2] True
- {"foo": "bar"} == {"foo": "bar"}

True

- // DİKKAT! TEHLİKELİ İŞLEMLER
- 1 is 1
- A = 2 ** 5
- B = 2 ** 5
- A is B
- json.loads('1') is json.loads('1')

is: referans eşitliği

- {"foo": "bar"} is {"foo": "bar"}
- A = 2 ** 10
- B = 2 ** 10
- A is B

False

- json.loads('10000000000000000') is json.loads('10000000000000000')

Python: Boolean olarak değerler

Truthy: Doğrumsu. Doğru olarak kabul edilebilen

Falsy: Yanlışımsı. Yanlış olarak kabul edilebilen

Boolean bağlamda (context) if içine yazıldığında False gibi davranan değerler:

Boş sayılar: 0, 0.0...

Boş diziler ve sözlükler: "", [ ], { }

None

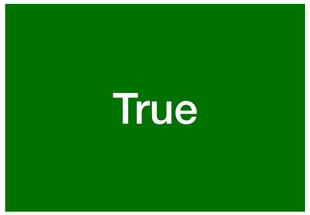
False

Şu şekilde genelleyebiliriz: Nesne türlerinin ilk değerleri falsy'dir.

JavaScript: Eşitlik, Denklik ve Tür Dönüşümleri

=== veya Object.is: değer ve referans eşitliği

- Daha çok pratikte bu tavsiye edilir
- İlkel değerler için değer eşitliği
  - 1 === 1
  - "Ustun" === "Ustun"
- Nesneler için referans eşitliği
  - [1, 2] !== [1, 2]
  - {"foo": "bar"} !== {"foo": "bar"}



==: tür zorlaması (type coercion) sonrası eşitlik

- Tehlikeli
- Kullanılması önerilmez
- 2 == "2"
- "" == 0
- null == undefined
- [ ] == ""

JavaScript: Nesneler İçin Değer Eşitliği

Henüz dil düzeyinde bir yöntem yok. Seçenekler:

- Nesnenin her veri öğesini tek tek ve özyinelemeli karşılaştıran bir metod yazmak
- JSON string değerine dönüştürüp karşılaştırmak
  - JSON.stringify([1,2]) === JSON.stringify([1,2])
- Bazı yardımcı kütüphanelerdeki metodları kullanmak
  - lodash.isEqual
  - assert.deepStrictEqual
  - immutable.is
  - expect(a).toEqual(b)

JavaScript: Boolean olarak değerler

Truthy: Doğru olarak kabul edilebilen

Falsy: Yanlış olarak kabul edilebilen

Boolean bağlamda (context) if içine yazıldığında false gibi davranan değerler:

false, 0, "", null, undefined

Şu şekilde genelleyebiliriz: İlkel türlerinin ilk değerleri falsy'dir.

DİKKAT: [ ] ve { } nesne türü olduğu için Python'daki gibi falsy değildir.

Aynı şekilde "false", " " gibi uzunluğu sıfırdan fazla olan stringler de doğrumsudur.

null

- Python'daki None gibi. Tek bir değer.
- let x = null
- x === null

- *Dikkat: typeof null == "object"*

undefined

var, let ve const ile bildirilmiş; ama henüz değer almamış değerler

```
let x
console.log(x)
x( )


-> x / undefined
is not a function
```

```
function bar( ) {
    console.log(x)
   // HATA!
  // ReferenceError:
  // x is not defined
}
```

```
function foo() {
    console.log(x)
   // Hata yok!
undefined
    var x = 10
}
```

undefined (devam)

Nesnelerde var olmayan alanlara ulaşıldığında

```
let x = {name: 'Ustun'}
console.log(x.ad) -> undefined
x.ad.toUpperCase() -> TypeError: Cannot read property 'toUpperCase' of undefined
x.getName() -> x.getName / undefined is not a function
```

undefined (devam)

1- var, let ve const ile bildirilmiş; ama henüz değer almamış değerler

let x

console.log(x)

x( ) -> x / undefined is not a function

2- nesnelerde var olmayan alanlara ulaşıldığında

Let x = {name: 'Ustun'}

x.getName() -> x.getName / undefined is not a function

3- fonksiyonlarda parametre olarak tanımlanan, ama argüman olarak verilmeyen değerler undefined olur.

Sayılar

İki tür: number ve BigInt türü

- Klasik integer türü yok. BigInt dışında bütün sayılar double-hassasiyetli (IEEE-754 64-bit ikili)
- 0.1 + 0.2 => 0.30000000000000004
- JavaScript'in suçu değil, Python'da da bu şekilde
- Trivia: maks. int 2^53-1 (Number.MAX_SAFE_INTEGER), 2^64 değil!
- Herhangi bir değeri sayıya dönüştürmek için Number() fonksiyonunu kullanın (ya da parseInt)
- new Number(x) değil. Sayı değil, sayı nesnesi oluşturur.

Sayılar

İki tür: number ve BigInt türü

- BigInt: Sayının sonuna n eklenirse BigInt (arbitrary precision - isteğe bağlı kesinlik)
- x = 1111111 1111111 1111111 1111111 1111111 1111111 1111111n
- y = 1111111 1111111 1111111 1111111 1111111 1111111 1111111n
- x + y
- Artık çoğu tarayıcıda destekleniyor (Safari'de TP'da)

String

- Ad = "Üstün"
- Aslında nesne değil, ilkel değer; ama String.prototype'taki metodları kullanabiliyor
- ad.toUpperCase()...
- Başka değerleri dönüştürmek için String(x) kullanın. New String(x) değil!
- 3 şekilde yazılabilir: Tek tırnak, çift tırnak, ters tırnak
- Çift ve tek tırnak Python'daki gibi
- Ters tırnak f-string'ler gibi.
- Aynı zamanda """ gibi son karakteri escape etmeden çok satıra izin vermekte.

String (devam)

Ters tırnak

```
ad = "Ali"

selam = `Merhaba ${ad}

Nasılsın?`
```

```
ad = "Ali"

selam = functionName`Merhaba ${ad}

Nasılsın?`
```

Tagged template:

functionName:
sanitization, html'e
dönüştürme...

Array & Object

- "Array literals and object literals[28] were inspired by similar features in the Python language." Wirfs-Brock & Eich.
- Python'daki gibi literaller. Liste yerine Array. dict ve class instance yerine Object.

```
sayilar = [1, 2, 3]

insan = {ad: 'Ali',
         soyad: 'Yilmaz',
         hobiler: ['Futbol','Muzik']}
```

Dikkat: Keyler string ya da Symbol türünde

Keylerin herhangi bir nesne olması için: new Map

Class ve instance oluşturma

```
class Ogrenci extends Insan {
    constructor(ad, soyad) {
        this.ad = ad
        this.soyad = soyad
    }

    merhabaDe(arkadas) {
         console.log(`Merhaba {arkadas}.
             Benim adim ${this.name} ve
             soyadim ${this.surname}`)
    }
}
```

```
let o = new Ogrenci("Ali", "Yilmaz")

o.merhabaDe("Ayse")
```

Python ile Farklar:

- extends sözcüğü
- __init__ yerine constructor
- Fonksiyonlarda explicit self yerine implicit this sözcüğü
- new sözcüğü

Fonksiyonlar

```
function foo(a, b) {
    console.log(a,b) }
```

```
foo(3,4) -> 3,4
foo(3) -> 3, undefined
foo(3,4,5) -> 3,4
```

```
def foo(a, b):
    print(a,b)

foo(3,4)

foo(3)
TypeError: foo() missing 1 required
positional argument: 'b'

foo(3,4,5)

TypeError: foo() takes 2 positional
arguments but 3 were given
```

Fonksiyonlar

Alternatif Fonksiyon Yazımları ve Türleri

```
let foo = function (a, b) {
    return a + b
}


let bar = (a, b) => { return a + b }

let bar2 = (a, b) => a + b
```

Array metodları ve list comprehensionlar

Python'daki list comprehensionlar yerine map/filter/reduce kullanılır

```
[1,2,3,4].map(x=>2*x)
```

```
[2* x for x in [1,2,3,4]]
```

```
[1,2,3,4].filter(x=>x % 2 == 0).map(x=>2*x)
```

```
[2* x for x in [1,2,3,4] if x % 2 == 0]
```

JavaScript'teki lambda fonksiyonlar, Python'dakinden daha güçlü, birden fazla ifade içerebilir.

Python lambdaları biraz da bilerek zayıf tasarlanmış, çünkü bu tarz lambdaları anlamak zorlaşabiliyor.

van Rossum'un ilgili makalesi:

https://www.artima.com/weblogs/viewpost.jsp?thread=98196

Destructuring ve spread: * ve ** yerine ...

spread: yaymak

```
[a, b] = [1, 2]

[a, b, ...c] = [1, 2, 3, 4, 5]

[...c, ...c] -> [3,4,5,3,4,5]


let kisi1 = {id: 1, ad: 'Ali'}

let {id, ad} = kisi1

let kisi2 = {id, ad, soyad: 'Ozgur'}

let kisi3= {  ...kisi2, soyad: 'Ozturk'}
```

```
[a, b] = [1, 2]

[a, b, *c] = [1, 2, 3, 4, 5]

[*c, *c] -> [3,4,5,3,4,5]


kisi1 = {'id': 1, 'ad': 'Ali'}

kisi2 = {**kisi1, 'soyad': 'Ozgur'}
```

Fonksiyon Parametrelerinde Varsayılan Değerler ve Destructuring

```
function foo(a, b=2) { return a + b}
foo(3,4)
foo(3)

function bar(a, b, ...gerisi) {
    console.log("gerisi", gerisi)
}

bar(3,4,5,6,7)
```

```
function baz(id, {ad, soyad}) {
   return `Merhaba ${id}. Ad ${ad}.
soyad ${soyad}`
}

baz(3,{ad: 'Ali', soyad: 'Ozgur'})

baz(3,{ad: 'Ali'})

baz(3) -> Hata: Cannot destructure
```

Fonksiyon Parametrelerinde Varsayılan Değerler ve Destructuring

```
function baz(id, {ad="Ali", soyad="Ozturk"} = { }) {
   return `Merhaba ${id}. Ad ${ad}. soyad ${soyad}`
}

> baz(3,{ad: 'Ali', soyad: 'Ozgur'})
'Merhaba 3. Ad Ali. soyad Ozgur'
> baz(3,{ad: 'Veli'})
'Merhaba 3. Ad Veli. soyad Ozturk'
> baz(3)
'Merhaba 3. Ad Ali. soyad Ozturk'
```

Modüller ve Paketler

Python

- Modül = Dosya
- Her şey diğer dosyalar tarafından import edilebilir
- _ ile başlarsa geleneksel olarak gizli *sayılır*
- Paket: Klasör şeklinde modül
  __init__.py veya init.py olmayan namespace package
- import foo denildiğinde sys.path'e bakar
- sys.path: şu anki klasör, Python'daki sistem klasörleri ya da virtual env'in klasörü
- İlk dosya modül değil, script olarak çalıştırılır.
- __name__ == "__main__"

JavaScript: Modüller ve Paketler

- Eskiden yoktu
- Web sayfalarında script etiketleriyle çağrılan modül yapısı
- İlk modül yapıları script'leri ardarda ekleyen yapılar şeklindeydi
- Google Closure compiler vb. objelerle kendi modül sistemlerini yaptılar
- node.js kendi modül sistemini yaptı
- ES2015 EcmaScript modüllerini standartlaştırdı.
- Mevcut durum: Henüz node.js doğrudan kararlı bir şekilde desteklemiyor. Deneysel modda ve farklılıklar mevcut.

node.js modülleri

- Her modül bir namespace
- Bütün değişkenlerin scope'u burada
- Global değişken strict modda yok
- Göreli ya da mutlak importlar
- Aynı proje ya da klasörde genelde göreli import yapılır, çünkü mevcut klasör NODE_PATH'e eklenmez.
- node_modules'taki modüller Python'daki gibi absolute import şeklinde import edilir.

node.js modülleri

- Göreli ya da mutlak importlar
- Aynı proje ya da klasörde genelde göreli import yapılır, çünkü mevcut klasör NODE_PATH'e eklenmez.
- node_modules'taki modüller Python'daki gibi absolute import şeklinde import edilir.

main.js

```
const lodash = require('lodash')

const {foo} = require('./bar')
```

bar.js

```
exports.foo = function foo() {

    ...

}
```

node.js modülleri

- Eğer o isimde bir JS dosyası yoksa o isimde bir klasör içindeki index.js import edilir.
- Bu açıdan Python'daki paketlere benzer. index.js == __init__.py

main.js

```
const lodash = require('lodash')

const {foo} = require('./bar')
```

bar/index.js

```
exports.foo = function foo() {

    ...

}
```

EcmaScript Modülleri

```
a.js'te

export function foo
```

```
Aynı klasördeki b.js'te

import {foo} from './foo'
```

node ve webpack benzeri bir bundler kullanırken node_modules'taki import'lar absolute gibi verilir.
Ör: node_modules klasöründeki lodash'ten random fonksiyonunu import etmek için:

import {random} from 'lodash'

pip ve virtual environment ile npm ve yarn

- pip yerine npm ya da yarn
- npm node ile birlikte geliyor
- npm install lodash
- Virtualenv yok, onun yerine node_modules klasörüne yükleniyor.
- Bu importları zorlaştırıyor, ama venv ile uğraşmaya gerek kalmıyor.
- pipenv ve poetry'deki gibi requirement lock (tüm bağımlılıkların kaydı) mevcut
- Bu durumda npm install yerine npm ci kullanılır.

Önemli Araçlar

- En yaygın araçlar:
  - Jest: test. pytest gibi
  - Webpack: bundler (birden fazla dosyayı birleştirmek)
  - Babel: yeni JavaScript özelliklerini eski JavaScript'e dönüştürmek için compiler/transpiler. 2to3 gibi
  - Eslint: Linter. pyflakes gibi
  - Prettier: Kod biçimlendirmek için. black gibi.
  - TypeScript, Flow: Tür denetimi yapmak için. mypy ve typings modülü gibi

Önemli Kütüphaneler

- Temel Web API
  - DOM, WebAudio, WebGL, Canvas...
- Bu API'lar üzerine inşa edilmiş kütüphaneler:
  - React, vue, jQuery, pixie

Sonuç: Python ve JavaScript: İki yakın dil

- Python JavaScript'e göre daha temiz bir dil; ancak her dilin artı ve eksileri var
- Python bilen biri JavaScript'i hızlıca öğrenebilir
- Kütüphaneleri öğrenmek işin daha önemli kısmı
- JavaScript'teki kütüphane sayısı daha çok, ortalama kütüphane kalitesi daha düşük
- Genelde büyük çatılar yerine küçük kütüphaneler daha yaygın.
- Ör: Django-Flask.

  JavaScript-Express. Önyüz tarafında React.
- Bilimsel hesaplamalar ve yapay zeka vb uygulamalar için Python daha zengin
- JavaScript her yerde
- Her iki dilin de geleceği parlak

Kaynaklar

- JavaScript: The First 20 Years, Wirfs-Brock, Allen; Eich, Brendan. https://zenodo.org/record/3707008
- Mozilla Developer Network: https://developer.mozilla.org/en-US/docs/Web
- ECMAScript-262 Standardı: https://tc39.es/ecma262
- Douglas Crockford'un makaleleri: https://www.crockford.com/javascript/

Dinlediğiniz için teşekkür ederim.

@ustunozgur
https://www.youtube.com/user/ustunozgur/videos
ustun at ustunozgur.com

Bu sunumun video kaydı: https://www.youtube.com/watch?v=EEXFcdTmgZM