

WEBPROGRAMOZÁS

KÖVETELMÉNYEK, JAVASCRIPT NYELVI ELEMEK

Horváth Győző

egyetemi docens

horvath.gyozo@inf.elte.hu

1117 Budapest, Pázmány Péter sétány 1/c., 2.408

Tel: (1) 372-2500/8469

DINAMIKUS WEBPROGRAMOZÁS

STATIKUS ↗ DINAMIKUS

EDDIGI ISMERETEK

Statikus weblapok készítése: **Webfejlesztés**

-  HTML5
-  CSS 1, 2, 3
-  Alapvető HTTP ismeretek
-  CSS keretrendszer (Bootstrap)

KLIENS-SZERVER ARCHITEKTÚRA

- **Web:** kliens és kiszolgáló kommunikációja
 - **HTTP:** a kommunikáció protokollja
 - **URL:** erőforrások azonosítója
 - **HTML:** dokumentumleíró nyelv
- Lépések
 1. Kliens kérést intéz a szervernek
 2. Szerver válaszol
 3. A kliens feldolgozza a választ

STATIKUS OLDALAK

- **Szerver szempontjából** statikus
 - Kérés pillanatában a szerveren megtalálható az a tartalom, amely leküldésre kerül a válaszban
 - Fájlkiszolgálás
- **Kliens szempontjából** statikus
 - A letöltött és a létrejött tartalom az oldal élettartamának a végéig ugyanaz
 - Nem változik meg sem a böngésző állapota, sem a betöltött dokumentum szerkezete
 - Nem fut le benne programkód, leíró nyelv, deklaratív

DINAMIKUS OLDALAK

- **Szerver szempontjából** dinamikus
 - A válaszként leküldött tartalmat program állítja elő
 - A kérés pillanatában a válasz még nem létezik a szerveren
- **Kliens szempontjából** dinamikus
 - A letöltött tartalomban programkód fut le
 - Ez megváltoztathatja a böngésző állapotát és a dokumentum szerkezetét
- ⇒ **PROGRAMOZÁS**

MIRŐL LESZ SZÓ?

- Platformspecifikus programozás
- Dinamikus **webprogramozás**
- Kliens-szerver architektúra minden oldalán
 - Kliensoldalon: **JavaScript**
 - Szerveroldalon: **PHP**
- Alapvető ismeretek, bevezetés
 - Új nyelvek
 - Új technológiák
 - Új programozási modellek

FÉLÉV FELÉPÍTÉSE

- I. Kliens oldali dinamikus webprogramozás: **JavaScript**
- II. Szerver oldali dinamikus webprogramozás: **PHP**

MOTIVÁCIÓ

KÖVETELMÉNYEK

INFORMÁCIÓK A TÁRGYRÓL

<http://canvas.elte.hu>

- Előadások
- Gyakorlati anyagok
- Beadandók
- ZH-k
- Eredmények

ÁLTALÁNOS INFORMÁCIÓK

- Óraszám: 1 előadás + 2 gyakorlat + 1 konzultáció
- Előfeltétel: Webfejlesztés
- Számonkérés: **folyamatos számonkérésű tárgy**

KÖVETELMÉNYEK (ÁLTALÁNOS)

- Előadás kötelező
- Gyakorlatok kötelezők (max. 3 hiányzás)
- Összesen **120 pont** szerezhető a félév során

KÖVETELMÉNYEK

- **Két csoport ZH**
 - **JavaScript:** 45 perc, 4. JS gyakorlat (10 pont)
 - **PHP:** 45 perc, 4. PHP gyakorlat (10 pont)
- **Két évfolyam ZH (pótolható)**
 - **JavaScript:** 2020.11.06. 16-18 (30 pont)
 - **PHP:** 2020.12.20. 16-18 (30 pont)
- **Két beadandó**
 - **JavaScript:** 2020.11.15. (20 pont)
 - **PHP:** 2021.01.10. (20 pont)

JEGYSZERZÉS FELTÉTELEI

- Részvétel az előadásokon
- Részvétel a gyakorlatokon
- Két elfogadott beadandó (legalább 40%)
- Két elfogadott évfolyam ZH (legalább 40%)

KLIENSOLDALI WEBPROGRAMOZÁS

JAVASCRIPT TÖRTÉNETE

- **1991:** világháló születése
- **1993:** első grafikus böngészők
- **1994:** Netscape Navigator böngésző
- **1995 áprilisa:** Netscape
 - Brendan Eich: olyan programozási nyelv, amelyekkel interaktívvá tehetők weboldalak
- **1995 decembere:** bejelentik a JavaScriptet

- **1996-1999:** I. böngészőháború  (Netscape vs IE)
 - A kliensoldal megbízhatatlan
- **1999-2004:** Sötét középkor
 - Szerveroldali technológiák fejlődése
- **2004-2017:** II. böngészőháború  (IE vs Firefox, Opera, ...)
- **2006-**: JavaScript újrafelfedezése (AJAX, jQuery)
- **2008:** Google Chrome
 - III. böngészőháború  (JS sebessége)
- **2009-**: parancssori JavaScript (Node.js)
- **2017:** Google Chrome  (Chromium)

NÉV ÉS SZABVÁNY

- Elnevezések:
 - LiveScript: eredeti név
 - JavaScript: marketing miatt a Java programozók átcsábítására (web Visual Basic-je)
- Szabvány: ECMAScript
 - Európai Informatikai és Kommunikációs Rendszerek Szabványosítási Szövetsége (ECMA)
- Microsoft
 - JScript

ECMASCRIPT VERZIÓK

- 1997: 1. verzió
- 1999: **3. verzió**
- 2009: **5. verzió (ES5)**
 - hibajavítások, apróbb fejlesztések
- 2015: **6. verzió (ES6, ECMAScript 2015)**
 - nagyobb fejlesztések, modern nyelvi tulajdonságok
- 2016: 7. verzió (ES7, ECMAScript 2016)
- 2017: 8. verzió (ES8, ECMAScript 2017)
- ...

Stages: 0, 1, 2, 3, 4 → szabvány

FEJLESZTŐESZKÖZÖK

**SZERKESZTŐK, BÖNGÉSZŐK, ESZKÖZTÁRAK,
DOKUMENTÁCIÓ**

SZERKESZTŐK

- Text editorok
 - Notepad++
 - Sublime Text 3
 - Atom
 - **Visual Studio Code**
- Integrált fejlesztőkörnyezetek (IDE)
 - NetBeans IDE
 - Visual Studio
 - WebStorm

Tetszőleges modern kódszerkesztő használható

BÖNGÉSZŐK

- **Google Chrome** (Blink)
- Mozilla Firefox (Gecko, Quantum)
- Microsoft Edge (Chromium, Blink)
- Opera (Chromium, Blink)
- Safari (Webkit)

Tetszőleges modern böngésző használható

Megjelenítő motorok kis történelme:

KHTML (Konqueror) → Webkit (Safari) → Blink (Chrome)

WEBFEJLESZTÉSI ESZKÖZÖK

- Webfejlesztő eszköztár (F12)
 - Elemek
 - JavaScript konzol (`console`)
 - Debugger
 - Erőforrások
 - Hálózati forgalom
 - Teljesítményprofilozás
- Parancssori eszközök

DOKUMENTÁCIÓ

- Hivatalos dokumentáció az **ECMAScript szabvány**
- Mozilla Developer Network: JavaScript
 - **Főoldal**
 - **JavaScript referencia**
 - **JavaScript guide**
 - Firefox specifikus dolgok, de a dokumentáció megfelelően jelzi

FUTTATÓ KÖRNYEZET

FUTÁSI KÖRNYEZET

A JavaScriptnek szüksége van egy futtató környezetre

- **Böngésző**
- Parancssor (Node.js)

HOVA ÍRHATJUK A KÓDOT?

- JavaScript konzolba (böngésző)
 - Az adott oldal kontextusában értelmezésre kerül
 - Kipróbálásra jó
- **HTML kódba** (böngésző)
 - Inline szkript, `<script>` tag, bárhova rakhatjuk
 - Külső állományba, `<script>` tag src attribútumával töltjük be
- Parancssori értelmezőbe (parancssor, Node.js)

JAVASCRIPT KÓD HELYE

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Webfejlesztés 2.</title>
    <script>
      //JavaScript code here
    </script>
    <script src="jscode.js"></script>
  </head>
  <body>
    <script>
      //JavaScript code here
    </script>
    <p>Hello world!</p>
    <script src="jscode2.js"></script>
  </body>
</html>
```

JAVASCRIPT KÓD HELYE

```
<!doctype html>
<html>
  <head>
    <!-- ... -->
  </head>
  <body>
    <!-- ... -->
    <script src="code.js"></script>
  </body>
</html>
```

A JAVASCRIPT NYELV

ALAPOK

MINTAKÉNT SZOLGÁLÓ NYELVEK

```
#include <iostream>
using namespace std;
int main() {
    //declaration
    const int n=5;
    double x[]={1,3,5,7,9};
    double s;
    //process
    s=0;
    for(int i=0; i<n; i++) {
        s=s+x[i];
    }
    //write output
    cout<<"Sum: "<<s<<endl;
    return 0;
}
```

C++

```
//declaration
const x=[1,3,5,7,9]
let s

//process
s=0
for(let i=0; i<x.length; i++) {
    s=s+x[i]
}
//write output
console.log('Sum: ', s)
```

JavaScript

C++

- Erősen típusos
- Fordított
- Általános célú programozási nyelv

JAVASCRIPT

- Dinamikusan típusos
- Interpretált
- Szkriptnyelv

DINAMIKUSAN TÍPUSOS

- A változók típusa a benne tárolt érték típusától függ
- Vagy másképp: a típusok az értékekhez tartoznak, nem a változókhöz.
- Automatikus típuskonverziók (lehetőleg kerüljük)

```
a = 'alma';
a = 12;
a == '12'; //true
```

INTERPRETÁLT

- A böngészőben futó értelmező értelmezi a JavaScript kódsorokat sorról sorra
- Nincs fordítási fázis, nem a lefordított kód fut
- Minimális előfeldolgozás történik
- A kód a hibáig lefut, ott elakad:
 - Az adott `<script>` blokk futása megáll
 - A `<script>` blokk után folytatódik az oldal betöltése

TOVÁBBI JELLEMZŐK

- Kis és nagybetűk különböznek (case sensitive)
- Nincs főprogram (`main`)
- Nincs input/output (csak API-kon keresztül)
- Nincs fájlkezelés (csak API-kon keresztül)
- Objektumorientált
- Prototípusos
- Automatikus pontosvessző beszúrás !!

```
let a = 12 // --> let a = 12;  
a = a + 1 // --> a = a + 1;
```

TÍPUSOK

Primitív értékek

null
undefined

Egyszerű típusok

Logikai	Boolean
Szám	Number
Szöveg	String

Összetett típusok

Objektum	Object
Tömb	Array
Függvény	Function

LITERÁLFORMÁK

Adott típus megjelenési formája

```
// Logikai Literál  
true  
false
```

```
// Szám Literál  
12  
12.34
```

```
// Szöveg Literál  
'Szöveg'  
"Szöveg"  
`Szöveg`  
  
'Tetszőleges ${kifejezés}'  
  
'Idézőjelben "így" macsakörmölök'  
"Macskakörömben 'így' idézek"  
'Idézőjelben \' idézőjel'  
"Macskakörömben \" macskaköröm"  
'Escape: \t \n \\ '
```

VÁLTOZÓK

- `var`, `let`, `const` kulcsszóval deklarálunk új változót
- ezek elhagyásával → globális változó – KERÜLENDŐ!!!
- Ha nincs kezdőérték → `undefined`

```
let nev = 'Győző'; // 'Győző'  
let masik;           //undefined
```

OPERÁTOROK

- Aritmetikai operátorok
 - `+`, `-`, `*`, `/`, `%`, `++`, `--`, unáris `-`, unáris `+`
- Értékadás operátorai
 - `=`, `*=`, `/=`, `%=`, `+=`, `-=`, stb.
- Összehasonlító operátor
 - `==`, `!=`, `==`, `!=`, `>`, `>=`, `<`, `<=`
 - `==` és `!=` típus és érték szerint
 - `==` és `!=` érték szerint (automatikus konverziók)

```
12 == '12' // true
12 === '12' // false
```

OPERÁTOROK

- Logikai operátorok
 - `&&`, `||`, `!`
- Szövegösszefűzés operátorai
 - `+`, `+=`
- Bitenkénti operátorok
 - `&`, `|`, `^`, `~`, `<<`, `>>`, `>>>`
- Speciális operátorok
 - `? :` feltételes operátor
 - `,` több kifejezés végrehajtása egy utasításban, visszatérési értéke az utolsó kifejezés

VEZÉRLÉSI SZERKEZETEK

```
// Elágazás
if (felt) {
    utasítások
}

if (felt) {
    utasítások
} else {
    utasítások
}
```

```
// Többirányú elágazás
switch(kifejezés) {
    case érték1:
        utasítások
        break;
    case érték2:
        utasítások
        break;
    default:
        utasítások
}
```

```
// Ciklusok
while (felt) {
    utasítások
}

do {
    utasítások
} while (felt);

for (let i=1; i<=n; i++) {
    utasítások
}
```

A JAVASCRIPT NYELV

FÜGGVÉNYEK

```
int factorial(int n) {  
    int f = 1;  
    for (int i=2; i<=n; i++) {  
        f *= i;  
    }  
    return f;  
}
```

C++

```
function factorial(n) {  
    let f = 1;  
    for (let i=2; i<=n; i++) {  
        f *= i;  
    }  
    return f;  
}
```

JavaScript

ALAPÉRTELMEZETT ÉRTÉKEK

```
// function declaration
function add(a, b = 3) {
    return a + b;
}

// function call
add(40, 2)      // 42
add(10)         // 13
add(50, 20, 10) // 70
add()           // NaN
```

LÉTREHOZÁSI FORMÁK

```
// function declaration
function add(a, b) {
    return a + b;
}

// function expression
const add = function (a, b) {
    return a + b;
}

// fat arrow
const add = (a, b) => {
    return a + b;
}
const add = (a, b) => a + b;
```

LITERÁLFORMA

Ahogy egy kifejezésben megjelenhet

```
function (a, b) {  
    return a + b;  
}  
  
// or  
  
(a, b) => a + b  
  
// for example  
const add = (a, b) => a + b
```

FÜGGVÉNY MINT PARAMÉTER

```
function countA(str) {
  let count = 0;
  for (const c of str) {
    if (c === 'a') {
      count++;
    }
  }
  return count;
}

console.log( countA("apple") ) // 1
```

FÜGGVÉNY MINT PARAMÉTER

```
function count(str, fn) {  
  let db = 0;  
  for (const c of str) {  
    if (fn(c)) {  
      db++;  
    }  
  }  
  return db;  
}  
  
console.log(  
  count("apple", c => c === 'a')  
)
```

A JAVASCRIPT NYELV

TÖMB

LÉTREHOZÁS, OLVASÁS

Literálforma: `[]`

```
// creation
const urestTomb = [];
const tomb = [12, 'alma', true];

// referencing an element
tomb[0]; // => 12;
tomb[1]; // => 'alma';
tomb[2]; // => true;

// length
tomb.length; // => 3
```

MÓDOSÍTÁS

```
const tomb = [12, 'alma', true];

// modification
tomb[0] = 13;

// new element at the end
tomb.push("new");

// new element somewhere (not recommended)
tomb[100] = 'far away';
tomb.length; // => 101
tomb[99]; // => undefined

// deleting (size remains the same)
delete tomb[1];
tomb[1]; // => undefined
tomb.length; // => 101
```

MÁTRIX

Tömbök tömbje

```
const m = [
  [1, 2, 3],
  [4, 5, 6],
];
m[1][2]; // => 6
```

ITERATÍV FELDOLGOZÁS

```
const gyumolcsok = [
  'alma',
  'korte',
  'szilva'
];

// A gyümölcsök kiírása a konzolra
for (let i = 0; i < gyumolcsok.length; i++) {
  console.log(gyumolcsok[i]);
}

// for..of ciklus (ES6)
for (const gyumolcs of gyumolcsok) {
  console.log(gyumolcs);
}
```

TÖMB MŰVELETEK

- `pop()`, `push(e)`, `shift()`, `unshift(e)`
 - végéről, végére, elejéről, elejére
- `reverse()`
 - megfordít
- `splice(honnan, mennyit)`
 - kivág
- `join(szeparátor)`
 - szöveggé fűz
- `indexOf(elem)`
 - keresés
- `includes(elem)`
 - eldöntés

```
const t = [1, 2, 3, 4, 5];

t.push(6);      // [1, 2, 3, 4, 5, 6]
t.pop();        // --> 6, [1, 2, 3, 4, 5]
t.unshift(0);   // [0, 1, 2, 3, 4, 5]
t.shift();      // --> 0, [1, 2, 3, 4, 5]
t.reverse();    // [5, 4, 3, 2, 1]
t.splice(2, 1); // [5, 4, 2, 1]
t.join('#');   // "5##4##2##1"
```

TÖMBFÜGGVÉNYEK

Programozási tételek megvalósítása

- `forEach`: általános ciklus
- `some`: eldöntés
- `every`: optimista eldöntés
- `map`: másolás
- `filter`: kiválogatás
- `reduce`: összegzés (sorozatszámítás)
- `find`: keresés (elem)
- `findIndex`: keresés (index)

PÉLDA – KIVÁLOGATÁS

```
const numbers = [1, 2, 3, 4, 5];

function filter(x, fn) {
  const out = [];
  for (const e of x) {
    if (fn(e)) {
      out.push(e);
    }
  }
  return out;
}
const evens = filter(numbers, e =>e % 2 === 0);

// instead

const evens = numbers.filter(e =>e % 2 === 0);
```

PÉLDA – ÖSSZEGZÉS

```
function sum(x) {  
    let s = 0;  
    for (const e of x) {  
        s = s + e;  
    }  
    return s;  
}  
  
// instead  
  
x.reduce((s, e) => s + e, 0);
```

DESTRUCTURING AND SPREAD

```
const numbers = [1, 2, 3, 4, 5];
const a = numbers[0];
const b = numbers[1];

// instead
const [a, b] = numbers;

// default values
const [a = 10, b = 20] = [100] // a:100, b:20

// rest
const [a, b, ...rest] = numbers; // --> rest:[3, 4, 5]

// swapping variables
[a, b] = [b, a]

// ignoring
const [a,,b] = numbers; // a:1, b:3

// spread
const a = [1, 2, 3];
const b = [a, ...10]; // b:[1, 2, 3, 10]
```

A JAVASCRIPT NYELV

OBJEKTUM

OBJEKTUM

- Kulcs-érték párok gyűjteménye
- Asszociatív tömbhöz hasonlít (hash)
- Rekord, osztálypéldány szimulálható
- JavaScriptben nagyon fontos szerepük van
- Majdnem minden objektum
- Ha az érték függvény → metódus

LÉTREHOZÁS, OLVASÁS

Literálforma: `{ }`

```
// creation
const uresObj = {};
const obj = {
  mezo1: 12,
  'mezo2': 'alma',
};

// referencing
obj.mezo1;      // => 12
obj['mezo1'];    // => 12
```

MÓDOSÍTÁS

```
const obj = {  
    mezo1: 12,  
    'mezo2': 'alma',  
};  
  
// modification  
obj.mezo2 = 'korte';  
  
// extending  
obj.mezo3 = true;  
  
// deletion  
delete obj.mezo1;  
obj.mezo1; // => undefined
```

METÓDUS (FÜGGVÉNY MINT ADATTAG)

```
const obj = {
  data: 42,
  metodus: function () {
    console.log('Foo: ', this.data);
  },
  metodus2() {
    console.log('Foo: ', this.data);
  }
};

obj.metodus();
obj.metodus2();
```

GETTER ÉS SETTER

```
const obj = {
  _data: 42,
  get data() {
    return _data;
  },
  set data(value) {
    _data = value;
  }
};

obj.data = 52;
obj.data; // 52
```

DINAMIKUS MEZŐNÉV

```
// Computed property names
const prop = 'foo';
const o = {
  [prop]: 'something',
  ['b' + 'ar']: 'new'
};

o.foo; // 'something'
o.bar; // 'new'
```

PÉLDÁK

```
//Tömb az objektumban
const zsofi = {
  kor: 7,
  kedvencEtelek: [
    'krumplipüré',
    'rántott hús',
    'tejberizs'
  ]
};
//Elem elérése
zsofi.kedvencEtelek[1];
// => 'rántott hús'
```

```
//Objektum az objektumban
const david = {
  kor: 4,
  cím: {
    iranyitoszam: '1241',
    varos: 'Budapest',
    utca: 'Egyszervolt utca',
    hazszam: 63
  }
};
//Elem elérése
david.cím.utca;
// => 'Egyszervolt utca'
```

FELDOLGOZÁS

```
const matyi = {
  kor: 1.5,
  fiu: true,
  cukis: true
}

// Feldolgozás a for..in ciklussal
for (const i in matyi) {
  console.log(i, matyi[i]);
}
// Eredmény
// => kor 1.5
// => fiu true
// => cukis true
```

ADATSZERKEZETEK MODELLEZÉSE

```
//C++ vector --> JS tömb
const kutyuk = [
    'telefon',
    'félhallgató',
    'pendrive',
    'e-könyv olvasó'
];
```

```
//C++ struct --> JS objektum
const hallgato = {
    nev: 'Mosolygó Napsugár',
    neptun: 'kod123',
    szak: 'Informatika BSc'
};
```

```
//Rekordok tömbje
const hallgatok = [
{
    nev: 'Mosolygó Napsugár',
    neptun: 'kod123',
    szak: 'Informatika BSc'
},
{
    nev: 'Kék Ibolya',
    neptun: 'kod456',
    szak: 'Informatika BSc'
];
];
```

ADATSZERKEZETEK MODELLEZÉSE

```
//Tömböt tartalmazó rekordok tömbje
const hallgatok = [
  {
    nev: 'Mosolygó Napsugár',
    neptun: 'kod123',
    szak: 'Informatika BSc',
    targyak: [
      'Programozás',
      'Webfejlesztés 2.',
      'Számítógépes alapismeretek'
    ]
  },
  {
    nev: 'Kék Ibolya',
    neptun: 'kod456',
    szak: 'Informatika BSc',
    targyak: [
      'Programozás',
      'Webfejlesztés 2.',
      'Diszkrét matematika',
      'Testnevelés'
    ]
  }
]
```

DESTRUCTURING AND SPREAD

```
const o = {  
  a: 42,  
  b: 28,  
}  
const a = o.a  
const b = o.b  
  
// instead  
const {a, b} = o;  
  
// renaming  
const {a: c, b: d} = o;  
  
// default values  
const {a = 10, b = 20} = {a: 42};  
const {a: c = 10, b: d = 20} = {a: 42};  
  
// rest  
const o = {  
  a: 42,  
  b: 28,  
  c: 12
```

DESTRUCTURING AND SPREAD

```
// nested objects
const david = {
  kor: 4,
  cim: {
    iranyitoszam: '1241',
    varos: 'Budapest',
    utca: 'Egyszervolt utca',
    hazszam: 63
  }
};
const { cim: { utca }} = david
```

class (ES6)

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
  // Getter
  get area() {
    return this.calcArea();
  }
  // Method
  calcArea() {
    return this.height * this.width;
  }
}

const square = new Rectangle(10, 10);

console.log(square.area); // 100
```

class – PUBLIKUS MEZŐK

```
class Product {
    name;
    tax = 0.2;
    basePrice = 0;
    price;

    constructor(name, basePrice) {
        this.name = name;
        this.basePrice = basePrice;
        this.price = (basePrice * (1 + this.tax)).toFixed(2);
    }
}
```

TESZTELÉS

PÉLDA

```
function add(a, b) {  
    return a + b;  
}
```

TESZTELÉS A KONZOLON

TESZTELÉS A KÓDBAN

```
function add(a, b) {  
    return a + b;  
}  
// console.assert  
console.assert(add(3, 2) === 5, '3 + 2 should be equal 5');  
console.assert(add(10, 0) === 10, '10 + 0 should be equal 10');
```

Click and check the console!

ÖSSZEFOGLALÁS

- C++ → JavaScript
- Adatszerkezetek
 - egyszerű: logikai, szám, szöveg
 - összetett: tömb, objektum, függvény
- Programozási tételek tömbfüggvényekként