Google Earth Engine kurzus 2019

A geoinformatikai adatbázisok kurzus keretén belül. Időpontok:

- ***október 29.***
- *november 5.*
- *november 12.*

Gulácsi András, SZTE
https://github.com/SalsaBoy990

Bevezetés – felhő alapú számítástechnika

- Rengeteg adat áll már rendelkezésünkre a világról – mindenféle okoseszközök, szenzorok, műholdak stb. gyűjtenek információt az élet minden területén, akár valós időben
- Ez az órási adatmennyiség a **BIG DATA**
- **3** fő jellemzője van (a 3V):
  - **MENNYISÉG** (volume)
  - **SEBESSÉG** (velocity)
  - **VÁLTOZATOSSÁG** (variety)
- Ennek a feldolgozása és elemzése hatalmas számítás- és erőforrásigényű → a felhő alapú számítástechnika létrejön

Bevezetés – felhő alapú számítástechnika

- A felhő alapú számítástechnika **hatalmas tárolókapacitást**, **nagy teljesítményű szervereket** és adatbázisokat kínál:
- *„A felhő alapú szolgáltatások (cloud computing) közös jellemzője, hogy a szolgáltatásokat nem egy dedikált hardvereszközön üzemeltetik, hanem a szolgáltató eszközein elosztva, a szolgáltatás üzemeltetési részleteit a felhasználótól elrejtve. Ezeket a szolgáltatásokat a felhasználók a hálózaton keresztül érhetik el.”*

A Google Earth Engine API

- A **GOOGLE EARTH ENGINE** egy felhő alapú számítási platform, amit műholdképek és egyéb földi megfigyelési adatok feldolgozására hoztak létre. Hozzáférést nyújt a Google képi adattáraihoz és biztosítja a számítási teljesítményt és a metódusokat a képek feldolgozására.
- A felhasználói hozzáférés egy vékony kliensen (webböngésző alkalmazás) keresztül történik
- Bemutató videó - jobban elmondja, mint én :)
  https://www.youtube.com/watch?v=4E6yQLoGO2o

A Google Earth Engine API

- Az **API**-n keresztül férünk hozzá az Earth Engine-hez és használunk a Google-fejlesztők által előre megírt vagy saját függvényeket, metódusokat, amelyekkel GIS elemzéseket végezhetünk el (API = alkalmazásprogramozási felület)
- Mi a **JavaScript** programozási nyelvet fogjuk használni a böngészőből
- A böngészőbe beépített **JavaScript motor** fordítja le a kódunkat és futtatja le. A Google felé küldünk el kéréseket a http protokollon keresztül, **JSON** formátumban (JavaScript Object Notation)

A Google Earth Engine API

- **TUTORIÁLOK, API KÉZIKÖNYV, ADATKATALÓGUS** érhető el a Google honlapján. Ezekben minden megtalálható a Google Earth Engine használatáról és az előre megírt függvényekről metódusokról.
- https://developers.google.com/earth-engine/
- Keresés az API kézikönyvben (gyorsabb így): *CTRL + f*

A JavaScript nyelv története

- *1995*: **Brendan Eich** fejlesztette ki 10 nap alatt! A World Wide Web egyik fő összetevője a HTML és a CSS mellett. És nem, nincs köze a Java-hoz!
- Webalkalmazások részét képezi, űrlapok validálására, a DOM manipulálására, interaktivitásra stb. használatos
- A JavaScript nagyon magas szintű interpretáló nyelv
- Dinamikus, gyengén típusos, prototípus alapú, több paradigmás, esemény alapú, automatikus szemétgyűjtéssel (garbage collector)
- Minden böngésző tartalmaz JS motort a szkriptek futtatásához

A JavaScript nyelv története

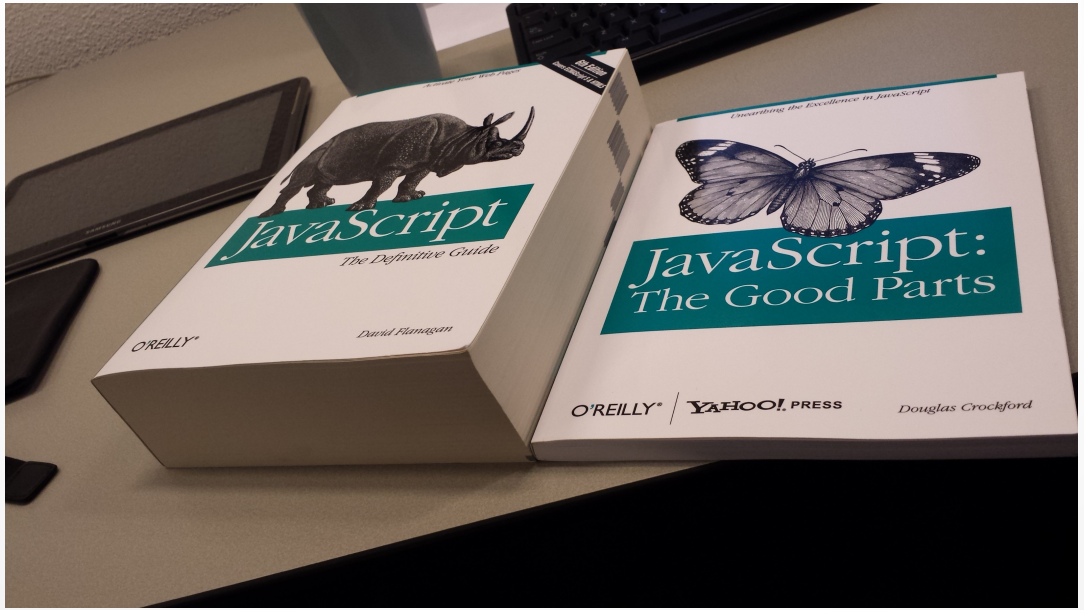
- The Birth & Death of JavaScript by Gary Bernhardt in 2014
- *2006*: **jQuery** könyvtár: a kliens oldali szkriptezés egyszerűsítésére (API)
- *2009*: **Node.js** megjelenése: aszinkron I/O könyvtárak egy JavaScript VM-mel párosítva, főként szerveralkalmazások írásához. JS futtatása shell/cmd-ben
- *2010*: **AngularJS** framework (Google), web app-ok
- *2013*: **React** framework megjelenése (Facebook), web app-ok
- *2016*: **Angular 2+** app platform (Google): TypeScript (TS) alapon

JavaScript a Google Earth Engine-ben

- A Google Earth Engine Kódszerkesztőben JavaScript-ben (JS) fogunk dolgozni és böngészőbe épített JS fordítómotor segítségével futtatjuk a szkriptjeinket
- Ehhez viszont tudni kellene programozni és ismerni a JavaScript-et.
- Ennek hiányában lehetetlen kihasználni a Google Earth Engine-ben rejlő lehetőségeket.

JavaScript – ajánlott tananyag

- Anthony Alicea 2015 - JavaScript: Understanding the Weird Parts, Udemy kurzus (19 €), 11,5 óra. https://www.youtube.com/watch?v=Bv_5Zv5c-Ts



A JavaScript (részleges) bemutatása

- A JS egyszálas (single-threaded), egyszerre csak egy parancsot hajt végre, sorban, szinkronban
- A böngésző nem feltétlenül így működik, vannak benne aszinkron részek is
- ES5 (ECMAScript 5) szabvány szerint programozunk
- A JS gyengén típusos nyelv, változókat és objektumokat a **var** kulcsszóval hozunk létre; a változó típusa változhat
- Az ún. primitív adattípusok: **undefined**, **null**, **sztring**, **logikai** és **szám** (beleértve a **NaN** értéket)

Primitív adattípusok

- A JS minden számot 64 bites lebegőpontos formátumban reprezentál (nincs külön integer típus), Unicode karakterkészlet
- `var a = 'alma';`
- `var b = true;`
- `var c = null;`
- `console.log(typeof NaN) // => number`
- `console.log(typeof undefined) // => undefined (típus)`
- `var d = 2;`
- `var π = 3.14;`

Primitív adattípusok

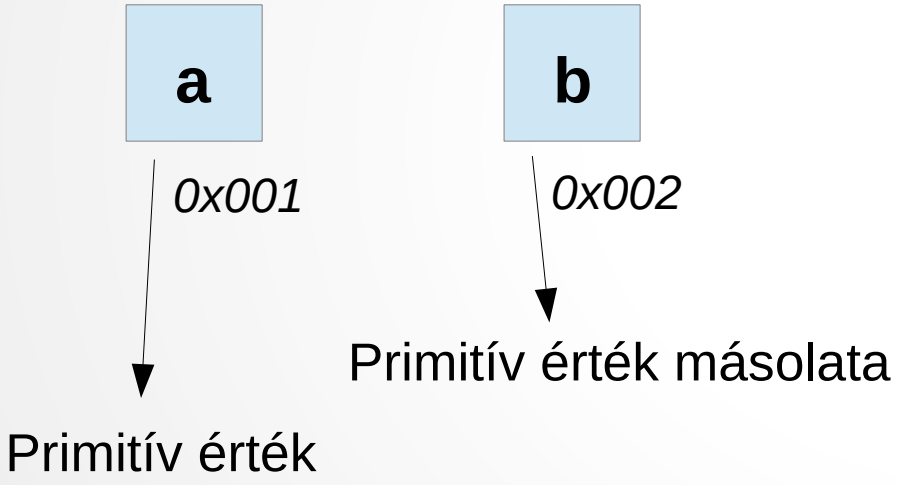
- A lebegőpontos számok nem egészen pontosak:
- `console.log(0.1 + 0.2); // => 0.30000000000000004`
- `console.log(0.1 + 0.2 === 0.3); // => false`
- A JS pontosan reprezentálja az egész számokat -9007199254740992 (-2^53) és 9007199254740992 (2^53) között
- A hamisértékek közé tartozik: a **false**, a **0**, a **null**, az **undefined**, az **""** (üres sztring) és a **NaN**.
- Ezek a feltételes kifejezésekben hamisnak értékelődnek ki

Primitív és referencia típus közti különbség

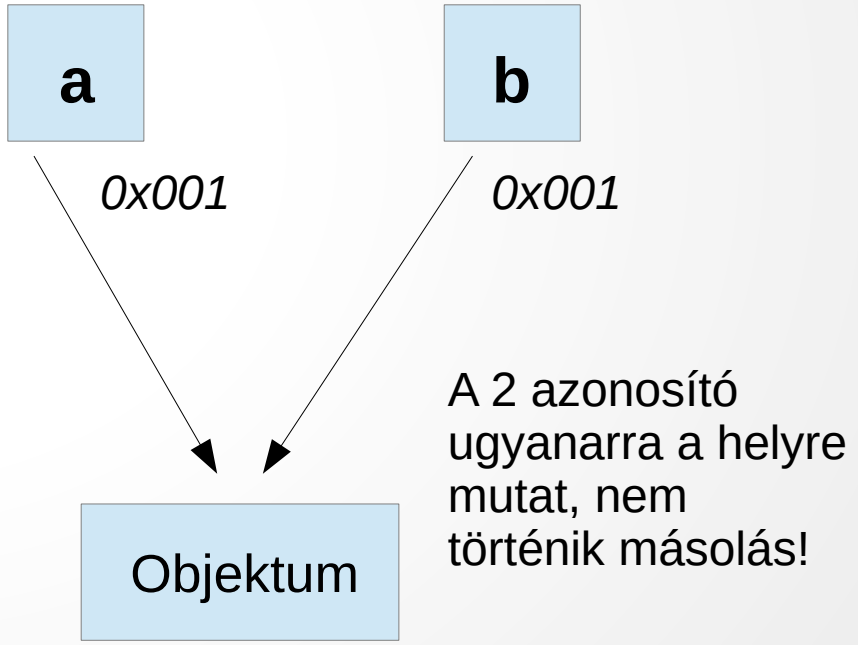
- `var a = 2;`
- `var b = a; // másolat készül`
- `b = 10;`
- `console.log(a); // => 2`
- `console.log(a === b); // => false`
- `var a = [1, 2, 3];`
- `var b = a;`
- `b[0] = 10;`
- `console.log(a); // => [10, 2, 3]`
- `// ugyanarra az objektumra hivatkoznak`
- `console.log(a === b); // => true`

Primitív és referencia típus közti különbség

```
b = a;
```




```
b = a;
```



Objektumok (referencia típusok)

- **OBJEKTUM = kulcs-érték (key-value) párok gyűjteménye:**

```
var hallgato = {
  nev: 'Béla',
  magyar: true,
  szuletesEve: 1111,
  szuletesnap: [1111, 11, 11],
  print: function () { console.log('Helló világ!') },
  cim: {
    varos: 'Szeged'
  }
};
```

Objektumok (referencia típusok)

```
// Tulajdonságok hozzáadása meglévő objektumhoz
hallgato.eletkor = 25;
hallgato['evfolyam'] = 2;
console.log('Életkor: ' + hallgato.eletkor);
console.log('Évfolyam: ' + hallgato.evfolyam);

console.log(hallgato.cim.varos);
console.table(hallgato.szuletesnap);

// []-kel is el lehet érni a tulajdonságokat
console.log(hallgato['nev']);
```

Globális objektum

- Ez egy szokványos JavaScript objektum, ami rendkívül fontos szerepet játszik. Közrefogja az összes JS kódunkat.
- A globális objektum tulajdonságai globálisan definiáltak, vagyis mindenhol hozzáférhetők (**globális hatókör**)
- A kliens oldali JavaScript-ben a **Window** objektum a böngészőablakban található összes JS kódunk globális objektumaként funkcionál. Pl. globális változókat a **window** objektum tagjaként hozzuk létre
- *console, undefined, parseInt(), Object(), JSON, Date(), Math, NaN*

Objektumok – objektum literál szintaxis

```
var person = {
  name: 'Jancsi',
  age: 33,
  print: function () {
    console.log(person.name + ' ' + person.age + ' éves');
  }
}
console.log(person);
console.table(person);
person.print();
```

Függvények

- A függvények is objektumok, változóknak értékül adhatók, függvények függvényt is visszaadhatnak
- A függvényeknek van kód tulajdonsága, ami az általunk megírt kódot tartalmazza

```
var square = function (x) {
  return x * x;
};
console.log(square.toString());
function square2 (x) {
  return x * x;
}
```

Függvény hatókör, hatókörlánc

- **Hatókör** = ahol a változók élnek, elérhetők, a saját végrehajtási környezetükben élnek (lásd köv. dia)
- JS-ben (ES5) **függvényhatókör** van (kivéve a try-catch blokkot).
- Minden változó, ami függvényen kívül található az globális: mindenhol elérhető.
- Az összes többi, ami a függvényeken belül van deklarálva, lokális, csak a függvényen belül él

Hatókör, hatókörlánc

```
var scope = 'globális'
function foo () {
  var scope = 'lokális'
  console.log(scope)
}
foo() // => lokális
console.log(scope) // => glob.
```

```
// DE MI TÖRTÉNIK EKKOR?
var scope = 'globális'
function foo2 () {
  scope = 'lokális'
  console.log(scope)
}
foo2() // => lokális
console.log(scope) // ??
```

Függvény függvényt is visszaadhat

```
function greet (lang) {
  if (lang === 'eng') {
    return function (name) {
      console.log('Hello ' + name + '!');
    }
  } else if (lang === 'esp') {
    return function (name) {
      console.log('Hola ' + name + '!');
    }
  } else { throw 'Unsupported language!'; }
}
var greetSpanish = greet('esp');
greetSpanish('János');
```

Hogyan fut le a kódunk?

- Először a JS motor létrehozza a böngészőablakunkban a **globális futtatási környezetet** (a globális objektumot, **Window**, és a **this** értéket, ami a globális objektumra hivatkozik).
- Ezután a **Syntax Parser** végigfut a kódunkon és megkeresi az összes **var**-t és **function**-t.
- A következő, létrehozási fázisban a JS motor **lefoglalja a memóriát** a változóink/objektumaink számára, értéküket ideiglenesen **undefined**-ra állítja (az értékeket majd később fogjuk definiálni).
- És csak ezután jön a **futtatási fázis**, amikor a JS motor végrehajtja az utasításokat sorban, egyesével.

Mi az a *this*?

- A **this** az éppen aktuális objektumpéldányra mutató referencia, amin keresztül hozzáférünk objektum tulajdonságaihoz.
- Mi nem fogjuk a **this**-t használni, ez a magyarázat csupán a metóduslánc megértéséhez kell

Objektumok – függvény („konstruktor”)

```
function Person (name, age) {
  this.name = name || 'John Doe';
  this.age = age || 0;
  this.print = function () {
    console.log('Person: ' + this.name + ', ' + this.age + ' éves');
  }
}
// példányosítás
var adam = new Person('Ádám', 25);
adam.print();
```

A metóduslánc

```
function Szam (ertek) {
  this.ertek = ertek;
  this.print = function () {
    console.log('A szám: ' + this.ertek);
  }
  // a következő dia kódja IDE jön //
}
```

A metódusLánc

```
// Négyzetre emelő függvény
this.square = function () {
  this.ertek *= this.ertek;
  return this;
}

// Negáló függvény
this.negate = function () {
  this.ertek = -this.ertek;
  return this;
}
```

```
var myNumber = new Szam(32);

myNumber.square().negate().print();
```

Funkcionális programozás

- **Mindent függvényekkel fejezünk ki**, amelyek egy bemenetet kimenetté alakítanak át
- Iterációk helyett is speciális függvényeket használunk (ennél természetesen sokkal többet jelent a funkcionális programozás, de itt elég ennyit tudni)
- `// operátor szintaxis`
- `var c = 3 + 5;`
- `// de fel lehet függvényként is fogni:`
- `function add (a, b) { return a + b; }`
- `c = add(3, 5);`

forEach – for ciklus helyett

```
[1, 2, 3].forEach(function (elem) {
  console.log(elem)
})

// for ciklussal ugyanez:
var arr = [1, 2, 3]
for (var i = 0; i < arr.length; i++) {
  console.log(arr[i]);
}
```

map – for ciklus helyett

```
var arr = [1, 2, 3]
function square (x) { return x * x }
var arr2 = arr.map(function (x) {
  return square(x)
});
console.table(arr2)

var arr3 = []
for (var i = 0; i < arr.length; i++) {
  arr3.push(square(arr[i]));
};          console.table(arr3)
```

reduce – példa: összeg számítása

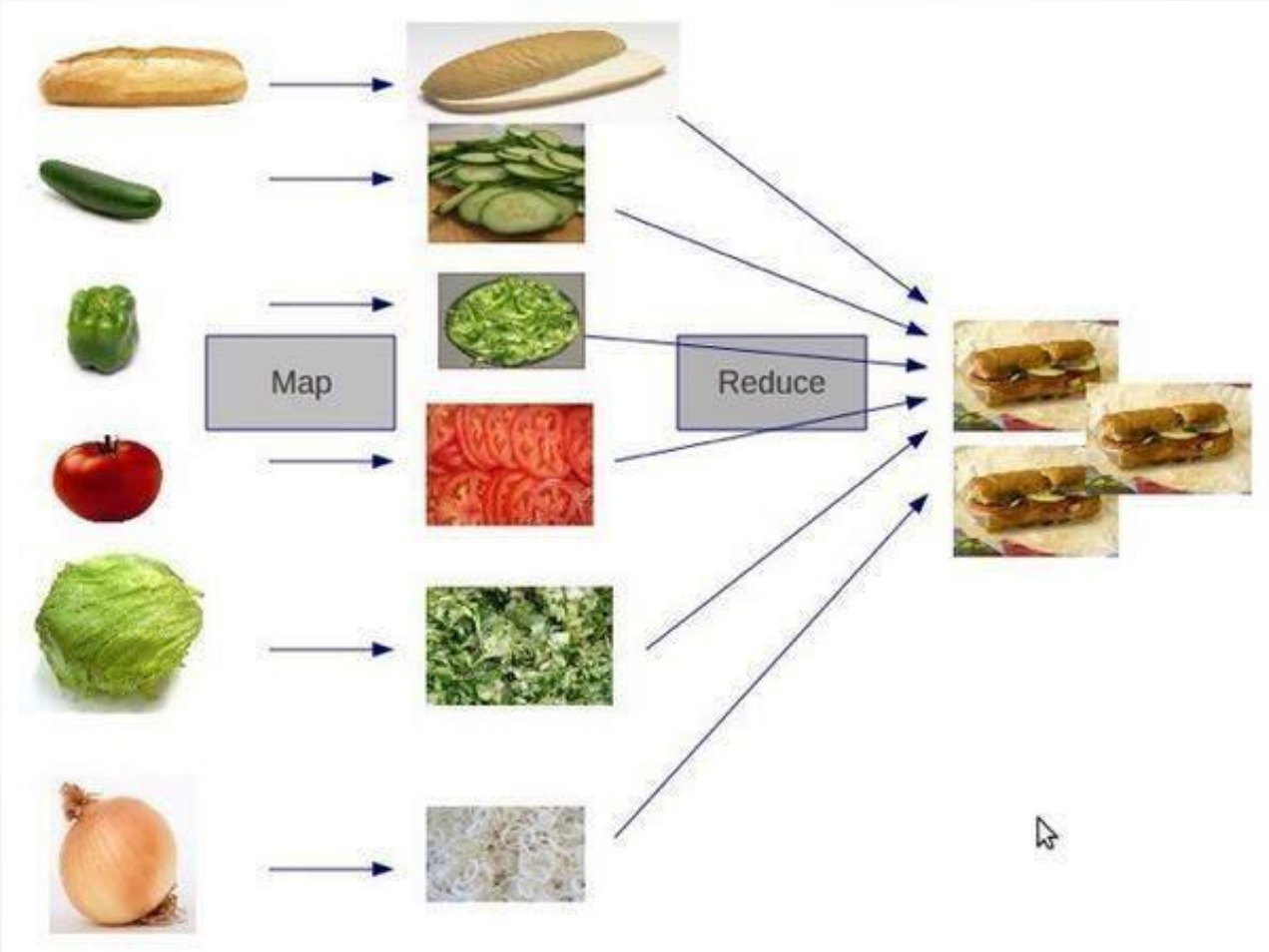
```
var osszeg = arr.reduce(function (accumulator, elem) {
  return accumulator + elem;
}, 0)
console.log(osszeg);

// Hagyományos módon:
var sum = 0;
for (var i = 0; i < arr.length; i++) {
  sum += arr[i];
}
console.log(sum);
```

filter – adatok szűrése

```
var arr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
var lessThanFive = arr.filter(function (elem) {
  return elem < 5;
});
console.table(lessThanFive)
var eredmeny = []
for (var i = 0; i < arr.length; i++) {
  if (arr[i] < 5) { eredmeny.push(arr[i]) }
}
console.table(eredmeny)
```

map(), reduce() és filter()



JSON (JavaScript Object Notation)

```
var obj = {
  x: 6,
  y: 13,
  a: 'abc',
  t: [1, 2, 3]
};

var json = JSON.stringify(obj)
console.log('JSON formátum: ' + json)
console.log('vissza objektummá: ' + JSON.parse(json))
```

===, !==, típuskonverziók, castolás

- A **változók típusa változhat** a futás során
- A JS-ben vannak **implicit** (rejtett) **típuskonverziók**, amik néha nehezen felfedezhető hibákat eredményezhetnek a kódodban, ha nem vagy tisztában velük.
- A **szigorú egyenlőség/nemegyenlőség** (===, !==) operátorokat használd a feltételes kifejezésekben! Két változó akkor egyenlő, ha értékük és típusok is egyezik. Más szóval a típuskonverziókat nem engedélyezik.
- **Castolás** = a változó típusának explicite konverziója, amire mi közvetlen utasítást adunk

===, !==, típuskonverziók, castolás

- `var szam = 101;`
- `var szoveg = ' kiskutya';`
- `var osszefuz = szam + szoveg;`
- `console.log({ osszefuz }); // => 101 kiskutya`
- `var bool = true`
- `(bool === true) ? console.log('Igaz!') : console.log('Hamis!');`
- `var x = '512' - 128; // => 384`
- `var y = '1500';`
- `y = Number(y); // => 1500`

A GIS-tudomány elmozdítása az automatizálás felé

- A Google Earth Engine lehetővé teszi a távérzékelési adatok gyors és majdnem teljesen automatizált feldolgozását.
- Gyorsan lehet több évtizednyi adatból idősorokat számolni, trendet vizsgálni, osztályozásokat, machine learning technikákat stb. alkalmazni.
- Manuálisan elég rossz lenne 3000 műholdképet egyenként feldolgozni kattintgatva az Erdasban

Néhány fontos tudnivaló indulás előtt

- A GEE nem kereskedelmi célra ingyenesen elérhető, viszont a használata az **Earth Engine Kiértékelők** (Earth Engine Evaluators) számára korlátozott
- A Google Earth Engine használatához Google Fiókkal kell rendelkezni: https://accounts.google.com/SignUp?hl=hu
- Egy űrlapot kell kitölteni itt: https://signup.earthengine.google.com/
- Kb. 1 hét alatt bírálják el, és adnak hozzáférést a platformhoz.

A Google Earth Engine használata általánosan

- Mindenki nyissa meg a GEE kódszerkesztőt, hogy együtt haladhassunk: https://code.earthengine.google.com/

1. *lépés: A mintaterület meghatározása*
2. *lépés: A képkollekció lekérdezése, a képkollekció szűrése hely, dátum, egyéni szűrő alapján, növekvő sorrendbe rendezése a felhőborítás szerint*
3. *lépés: Az első kép kiválasztása és megjelenítése a térképen*
4. *lépés: A mintaterület kivágása a képből*
5. *lépés: A kép exportálása GeoTiff-be, a Google Drive-ra*

1. lépés: A mintaterület meghatározása

- **Rajzolással** a Google Maps térképen: poligon eszköz
- **Téglalappal**:
- `var region = ee.Geometry.Rectangle(19.1223, 46.7513, 19.2341, 46.8884);`
- **Vektoros adatokkal** (shape fájl, WGS84-ben!)
- `var studyArea = ee.FeatureCollection('users/gulandras90/shapefiles/study_area');`
- A mintaterület megjelenítése a térképen:

1. lépés: A mintaterület meghatározása

```
Map.centerObject(studyArea);
Map.setCenter(19.072, 47.204, 7);

// Outline for the study area
Map.addLayer(ee.Image().paint(studyArea, 0, 2), {},
'Study Area');
```

2. lépés: A képkollekció lekérdezése

```
// Cloud percentage property for filtering
var cloudFilter = ee.Filter.lessThanOrEquals('CLOUD_COVER', 20);
// FILTER Landsat COLLECTION
var filteredCollection =
ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
  .filterBounds(studyArea)
  .filterDate('2018-06-01', '2018-10-31')
  .filter(cloudFilter)
  .sort('CLOUD_COVER', false)
  ;
// Print filtered images
print(filteredCollection);
```

3. lépés: Az első kép kiválasztása és megjelenítése

```
// A legjobb minőségű kép kiválasztása a felhőborítás mértéke alapján
var first = filteredCollection.first();
print(first);

var vizParams = {
  bands: ['B4', 'B3', 'B2'],
  min: 547,
  max: 3408,
  gamma: 1.3
};
Map.addLayer(first, vizParams, 'Landsat 8 false color');
```

4. lépés: A mintaterület kivágása

- `var firstClipped = first.clip(studyArea);`
- `Map.addLayer(firstClipped, vizParams, 'Landsat false color clipped');`

5. lépés: A kép exportálása GeoTiff-be

```
// Save the composite image as GeoTIFF
Export.image.toDrive({
    image: firstClipped.select(
                ['B1','B2','B3','B4','B5', 'B7']),
    description:  'L8_miklapuszta_' + '2018',
    scale: 30,
    region: studyArea
});
```

HÁZI FELADAT

- Válassz ki egy tetszőleges mintaterületet, és készíts egy kivágatot egy Landsat képből (surface reflectance)! Az eredményképeket mentsd le a Google Drive-ra!

a) *Elsőnek rajzold körbe a területet a poligon eszközzel, és arra vágd ki a képet!*

b) *Most egy shape fájlt használj! Ha szükséges, akkor transzformáld WGS84 vetületi rendszerbe, mert csak ezt fogadja el a Google Earth Engine. És erre vágd ki a képet!*

- A mintaterületet a **miklapuszta.shp** fájl tartalmazza!