8. hét / Prológus

A mai órán a következőkről lesz szó:

- Elméleti bevezetés (~50-60 perc)
- Alapvető gépi tanulási algoritmusok (~25-30 perc)

8.hét / I. Elméleti bevezetés a mesterséges intelligenciába

Bevezetés a mesterséges intelligenciába

Maga a **mesetерséges intelligencia** (MI, vagy *AI*) a 21. század legfontosabb számítástudományi ágazata. A definiálása viszont koránt sem triviális! Ennek a legfőbb oka, hogy maga az *intelligencia* egy borzalmasan nehezen megfogalmazható, és értelmezhető fogalom - viszont ennek definiálása már inkább a filozófusok és legfőképpen a jogászok feladata.

A mi tárgyalásunkban sokkal célszerűbb a mesterséges intelligenciát, mint bizonyos algoritmusok, programozási problémák, paradigmák, megközelítések halmazaként definiálni! Széleskörben elfogadott, hogy például a gépi látás feladatai - amiket az előző két tanóra alkalmával sorra vettünk - a mesterséges intelligencia halmazába besorolhatóak. Más algoritmusok esetében ennél gyakran önkényesebb és kevésbé egyértelmű ennek meghatározása.

Mivel a mesterséges intelligencia fogalma exponenciálisan növekedik, ezért fontos már itt kiemelni, hogy a kurzus során kizárólag a **gépi tanulás**sal, illetve azon belül is elsősorban a **neurális hálózatok**kal fogunk foglalkozni. Erre többek között azért van szükség, mert a mesterséges intelligenciai algoritmusok jelentős részének vagy csak nagyon korlátozott mértékben, vagy már egyáltalán nincs is jelentősége a 2020-as években. Az AI egy páratlanul dinamikusan fejlődő tudományág és a történelem során gyakran előfordult, hogy bizonyos algoritmusok kikoptak a repetoárból, mert megjelent egy-egy lényegesen jobb megoldás.

Paradigmaváltás: Machine Learning

Eddig minden programozási problémát úgynevezett **algoritmikus módon** közelítettünk meg. A rendszerünk bemenetét és kimenetét három építőkocka segítségével próbáltuk meg összekötni, név szerint:

1. Utasításokkal és **függvények**kel;
2. Feltételes **elágazások**kal, vagy *if statement*-ekkel;
3. **Ciklusok**kal, vagy *loop*-okkal.

Ezek definiálják az úgynevezett **imperatív** (1.) és **strukturált programozás** (2. és 3.) paradigmákat. Később ennek kiterjesztéseként ismerkedtünk meg az **objektum-orientált programozás** fogalmával, amely ezen három építőkocka tetejére emelt egy újabbat: az *objektum*ot. Az objektum pedig egységbe zárja - ha úgy tetszik maga alá rendeli - a másik három alapegységet.

Ezzel szemben az *adattudomány*ok egy teljesen más szemlélet alapján gondolkodnak. Egy machine learning algoritmus ezzel szemben azt mondja, hogy a rendszer bemenete és közötti

függvénykapcsolatot (a korábban bemutatott építőkockák helyett) maguk az adatok definiálják. A rendszer felépítésének középpontjába a **tanulás**t és **tanítás**t helyezik, amely nem más, mint a bemenet és kimenet közötti (matematikai) függvény közelítése és megkeresése.

Az alapötlet az, hogy ameddig a bemenet és minden bementre adott kimenet egyértelmű, addig a kettőt egymáshoz rendelő függvények és rendszernek is egyértelműnek kell lennie. Ha pedig ez a függvénykapcsolat egyértelmű, akkor matematikai, statisztikai, valószínűségszámítási módszerekkel meg is tudjuk ezeket keresni. A tanítási folyamat előrehaladtával jellemzően javul a modellünk, viszont előfordulhat az is, hogy túltanul a modellünk. Erről a neurális hálózatok tárgyalásakor bővebben visszatérünk.

A legfontosabb különbség, hogy míg az algoritmikus megközelítés során minden utasítás, elágazás és ciklus előre definiált - függetlenül attól, hogy egy megadott bemenetre helyes kimenetet ad-e avagy sem. Ezzel szemben a gépi tanulás módszerei kizárólag a bemenő adatokból építi fel magát az algoritmust, az adatbázis alapján felismer abban szabályszerűségeket, mintázatokat, (gyakran olyanokat is, amelyeket emberi programozók nehézkesen, vagy egyáltalán nem tudtak volna leírni), és ezek alapján egy közelítő modellt ad.

Machine Learning feladatok

A gépi tanulás által megoldani kívánt problémák közül kettőt tárgyalunk mélyebben:

- **Klasszifikáció**: ismeretlen adatpontok besorolása különböző, jól elkülöníthető, *diszkrét* kategóriákba. Ilyen problémák megoldása során jellemzően egy matematikai függvényt keresünk, amely megfelelő módon felszabdalja az adathalmazunkat diszkrét osztályokra. Például ide tartoznak az orvosi diagnosztikai feladatok.
- **Regresszió**: olyan folyamat, melynek során kapcsolatot keresünk az úgynevezett független és függő változó között (mint input és output között), hogy ismeretlen adatokra jóslatokat, *predikciókat* tegyünk. Például ide sorolható az időjárás-előrelejzés.

Ezen felül még a gépi tanulás problémái alá tartoznak az alábbi problémák is:

- Klaszterezés;
- Dimenziócsökkentés;
- Anomáliadetektálás;
- Generatív modellezés.

A tanítási módok osztályozása

Azok alapján, hogy tanítás során figyelembe vesszük-e az adatokhoz rendelt kimenetet, avagy sem két fő tanítási paradigmát különböztetünk meg:

- **Supervised learning** (felügyelt tanítás): a modell a bemeneti adatok és a hozzájuk rendelt kimenetek (*label*-ök) alapján tanul. Ilyenkor a modell tanításának célja, hogy predikciókat tegyen ismeretlen adatokra vonatkozóan. Például ide tartozik a kép- és beszédfelismerés, orvosi diagnosztika, gépi fordítás.

- **Unsupervised learning** (felügyelet nélküli tanítás): a modell számára tanítás során kizárólag a bemenet elérhető. Ebben az esetben a tantás célja, hogy az adatokban belső mintázatokat, jellegzetességeket (feature) és összefüggéseket találjon a modell. Ennek segítségével tudjuk az adatunkat csoportosítani (clustering) vagy a paramétertér dimenzióját csökkenteni (dimensionality reduction). Például ide sorolhatóak a klaszterezések.

Ezen felül még a tanítási paradigmák közé tarozik az úgynevezett **Reinforcement Learning** (megerősítéses tanulás) is, amelynek lényege, hogy a betanítani kívánt ágens egy megadott környezetben folyamatosan döntéseket hoz, hogy egy úgynevezett *reward function*, (jutalmazási függvény) alapján maximalizálja a teljesítményét. Az ágens folyamatosan tanul a korábbi döntéesiből, és a büntető, illetve jutalmazó visszajelzések alapján halad a tanítás. Ezen paradigmát gyakran alkalmazzák robotikában vagy autonóm járművek esetén.

Machine Learning feladatok megoldása

Jellemzően a gépi tanulás feladatai az alábbi lépésekből épülnek fel:

1. **Adatgyűjtés**, mérés: Az adattudományok nem fognak működni adatok nélkül! Fontos, hogy a mérések elvégzése előtt tegyünk becsléseket arra vonatkozóan, hogy mennyire sok adatra van szükségünk és mekkora paramétertérrel akarunk dolgozni! Minél több paramétert mérünk, annál több adatra van szükségünk! (A paramétertér dimenziójával exponenciálisan növekszik a szükséges adattér "térfogata".)

2. **Adatfeltérképezés**: alapvető - statisztikai eszközökkel történő - megismerkedés az adatbázisunkkal. Például ábrázoljuk az egyes attribútumok eloszlását, szórását, vagy kirajzoljuk a térbeli és időbeli folyamatokat.

3. ! **Adatok előkészítése**, data preprocessing: az adathalmaz előkészítése a modellek betanításának megfelelően. Ez maga a "kreatív agya" az adattudományoknak, amikor a nyers adatbázisból kiszűrjük a számunkra lényeges részeket. Az adat minőségi paramétereitől függően teljesen eltérő eszközöket igényel: például egy képet gyakran fekete-fehérre átállítunk, átméretezünk, megkeressük és kiemeljük rajta az éleket, hogy kizárólag a lényeges információtartalommal rendelkező adattal dolgozzon az algoritmusunk. Minél ügyesebben tudjuk elvégezni az adathalmaz előkészítését, annál több nehézségtől mentjük meg magunkat meg a tanítás során!

4. **Modellválasztás**: a tanító modell meghatározása. (*Megjegyzés: valójában a modellt ismernünk kell már a preprocessing előtt, elvégre a választott modellre fogjuk optimalizálni a nyers adathalmazt!*) Ez mindig azzal kezdődik, hogy definiáljuk a problémát (például klasszifikációt, vagy regressziót akarunk-e csinálni) és ennek megfelelően keresünk a szakirodalomban megfelelő megoldásokat. Ezek alapján összeállítjuk a modellünket.

5. **Modell illesztés**, tanítás: a választott modell betanítása.

6. **Kiértékelés**: a tanító folyamat kiértékelése megadott teljesítménymetrikák (például *precision*, *recall*, vagy *accuracy*) alapján. Érdemes ezek változását is ábrázolni a tanítás során.

7. **Modell paramétereinek finomhangolása**: az előfeldolgozás, vagy a választott modellünk módosítása a teljesítménymetrikák szerint; illetve a tanítás során fellépő rendellenességek és anomáliák feltérképezése. Gyakran előfordul, hogy ugyan működik is a modellünk, de esetleg nem elég pontos, nem elég megbízható, nem tud eléggé általánosítani, túltanult. Ilyenkor az első négy lépésen visszafelé érdemes elkezdeni végighaladni és onnan újrakezdve optimalizálni a modellt.

8. **Alkalmazás**: az előre meghatározott célnak megfelelő modell telepítése.

8. hét / II. Alapvető ML algoritmusok

Lineáris regresszió

A lineáris regresszió egyike a *supervised* tanulási algoritmusoknak, amellyel regressziós problémákat lehet megoldani. Ezen legegyszerűbb modell egy $\mathbf{y} = \mathbf{A}\mathbf{x}$ lineáris egyenes segítségével közelíti az adathalmazt, ahol:

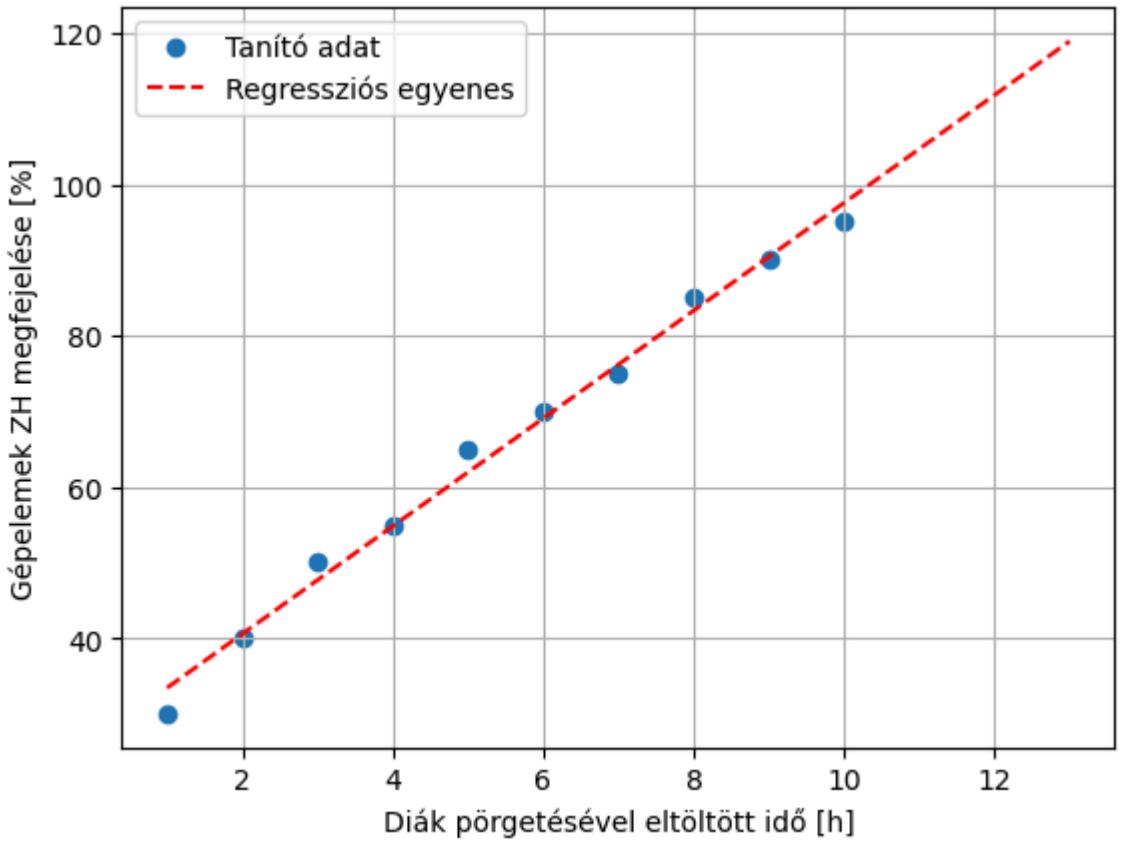
- $\mathbf{x}$ *a független változó*: amely alapján szeretnénk predikciókat tenni (INPUT);
- $\mathbf{y}$ *a függő változó*: amelynek az értékére szeretnénk predikciókat tenni (OUTPUT);
- $\mathbf{A}$ *a modell*.

Példa: Gépelemek ZH

A megadott eredmények alapján próbáljuk megjósolni, hogy ha többet tanultunk volna, akkor vajon megúszhattuk volna, avagy sem a pót ZH-t?

In [ ]:
```
%pip install scikit-learn
```

In [ ]:
```
# Szükséges könyvtárak
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Adathalmaz: tanulási idő és osztályzatok kapcsolata
study_hours = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]).reshape(-1, 1)
exam_scores = np.array([30, 40, 50, 55, 65, 70, 75, 85, 90, 95])

# Modellválasztás
model = LinearRegression()

# Modell illesztése
model.fit(study_hours, exam_scores)

# Kiértékelés
new_hours = np.array([1, 11, 12, 13]).reshape(-1, 1)
predicted_scores = model.predict(new_hours)

# Plotolás
plt.scatter(study_hours, exam_scores, label='Tanító adat')
plt.plot(new_hours, predicted_scores, color='red', linestyle='--', label='Regressziós egyenes
plt.xlabel('Diák pörgetésével eltöltött idő [h]')
plt.ylabel('Gépelemek ZH megfejelése [%]')
plt.grid()
plt.legend()
plt.show()
```



Klaszterezések

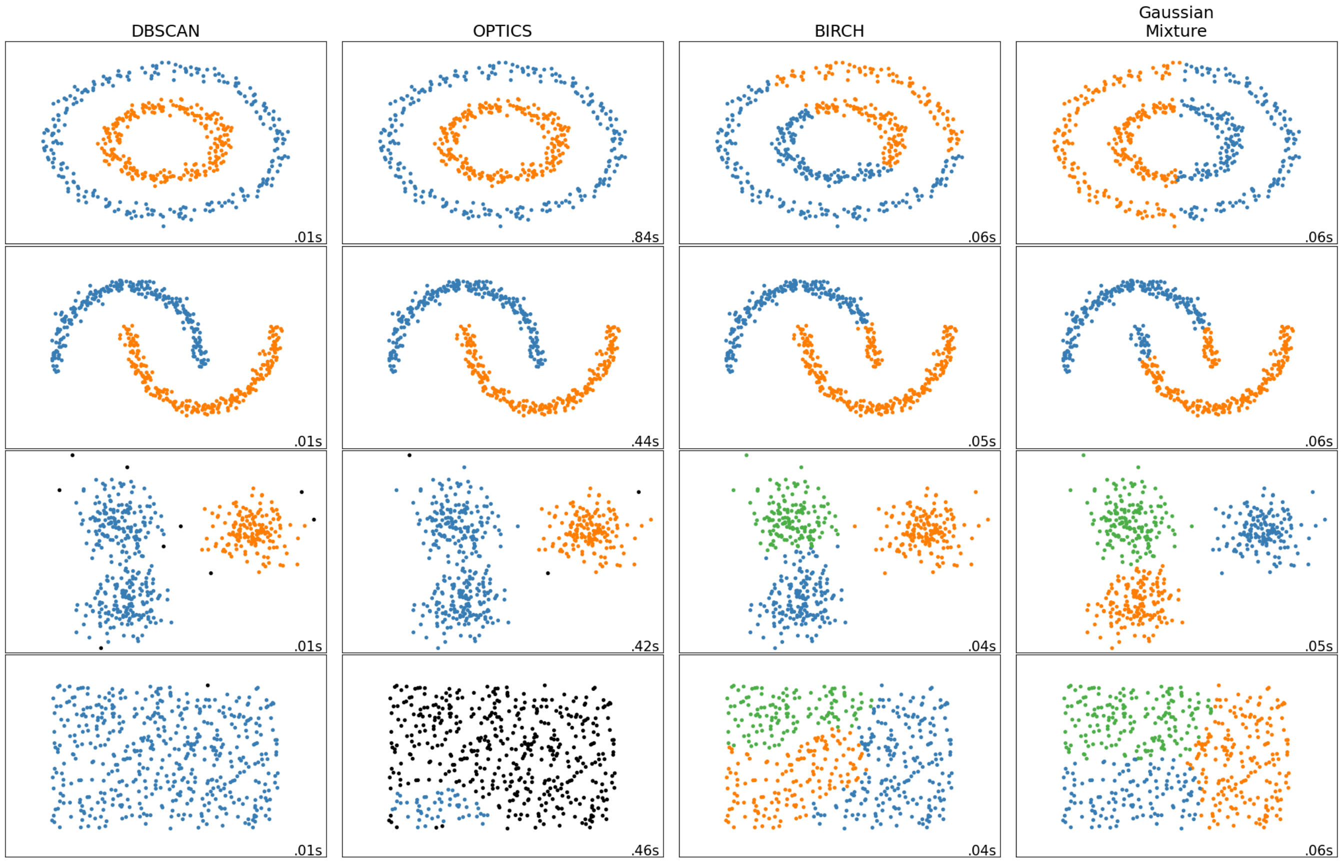
In [ ]:

```
import time
import warnings
from itertools import cycle, islice

import matplotlib.pyplot as plt
import numpy as np

from sklearn import cluster, datasets, mixture
from sklearn.preprocessing import StandardScaler

# Adathalmazok
n_samples = 500
seed = 30

noisy_circles = datasets.make_circles(
    n_samples=n_samples, factor=0.5, noise=0.05, random_state=seed
)
noisy_moons = datasets.make_moons(n_samples=n_samples, noise=0.05, random_state=seed)
blobs = datasets.make_blobs(n_samples=n_samples, random_state=seed)
no_structure = rng.rand(n_samples, 2), None

# Klaszterezés paraméterei
plt.figure(figsize=(21, 13))
plt.subplots_adjust(
    left=0.02, right=0.98, bottom=0.001, top=0.95, wspace=0.05, hspace=0.01
)

plot_num = 1

default_base = {
    "quantile": 0.3,
    "eps": 0.3,
    "damping": 0.9,
    "preference": -200,
    "n_neighbors": 3,
    "n_clusters": 3,
    "min_samples": 7,
    "xi": 0.05,
```

```
    "min_cluster_size": 0.1,
    "allow_single_cluster": True,
    "random_state": 42,
}

datasets = [
    (
        noisy_circles,
        {
            "damping": 0.77,
            "preference": -240,
            "quantile": 0.2,
            "n_clusters": 2,
            "min_samples": 7,
            "xi": 0.08,
        },
    ),
    (
        noisy_moons,
        {
            "damping": 0.75,
            "preference": -220,
            "n_clusters": 2,
            "min_samples": 7,
            "xi": 0.1,
        },
    ),
    (blobs, {"min_samples": 7, "xi": 0.1, "min_cluster_size": 0.2}),
    (no_structure, {}),
]

for i_dataset, (dataset, algo_params) in enumerate(datasets):
    # update parameters with dataset-specific values
    params = default_base.copy()
    params.update(algo_params)

    X, y = dataset

    # normalize dataset for easier parameter selection
    X = StandardScaler().fit_transform(X)

    # Modellválasztás
    dbscan = cluster.DBSCAN(eps=params["eps"])
    optics = cluster.OPTICS(
        min_samples=params["min_samples"],
        xi=params["xi"],
        min_cluster_size=params["min_cluster_size"],
    )
    birch = cluster.Birch(n_clusters=params["n_clusters"])
    gmm = mixture.GaussianMixture(
        n_components=params["n_clusters"],
        covariance_type="full",
        random_state=params["random_state"],
    )

    clustering_algorithms = (
        ("DBSCAN", dbscan),
        ("OPTICS", optics),
        ("BIRCH", birch),
        ("Gaussian\nMixture", gmm),
    )

    # Kiértékelés
    for name, algorithm in clustering_algorithms:
        t0 = time.time()

        # catch warnings related to kneighbors_graph
        with warnings.catch_warnings():
            warnings.filterwarnings(
```

```
                "ignore",
                message="the number of connected components of the "
                + "connectivity matrix is [0-9]{1,2}"
                + " > 1. Completing it to avoid stopping the tree early.",
                category=UserWarning,
            )
            warnings.filterwarnings(
                "ignore",
                message="Graph is not fully connected, spectral embedding"
                + " may not work as expected.",
                category=UserWarning,
            )
            algorithm.fit(X)

        t1 = time.time()

        # Predikciók
        if hasattr(algorithm, "labels_"):
            y_pred = algorithm.labels_.astype(int)
        else:
            y_pred = algorithm.predict(X)

        plt.subplot(len(datasets), len(clustering_algorithms), plot_num)
        if i_dataset == 0:
            plt.title(name, size=18)

        colors = np.array(
            list(
                islice(
                    cycle(
                        [
                            "#377eb8",
                            "#ff7f00",
                            "#4daf4a",
                            "#f781bf",
                            "#a65628",
                            "#984ea3",
                            "#999999",
                            "#e41a1c",
                            "#dede00",
                        ]
                    ),
                    int(max(y_pred) + 1),
                )
            )
        )
        # add black color for outliers (if any)
        colors = np.append(colors, ["#000000"])
        plt.scatter(X[:, 0], X[:, 1], s=10, color=colors[y_pred])

        plt.xlim(-2.5, 2.5)
        plt.ylim(-2.5, 2.5)
        plt.xticks(())
        plt.yticks(())
        plt.text(
            0.99,
            0.01,
            ("%.2fs" % (t1 - t0)).lstrip("0"),
            transform=plt.gca().transAxes,
            size=15,
            horizontalalignment="right",
        )
        plot_num += 1

plt.show()
```

```
c:\Users\mbenc\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: KMea
ns is known to have a memory leak on Windows with MKL, when there are less chunks than availa
ble threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\mbenc\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: KMea
ns is known to have a memory leak on Windows with MKL, when there are less chunks than availa
ble threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\mbenc\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: KMea
ns is known to have a memory leak on Windows with MKL, when there are less chunks than availa
ble threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\mbenc\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: KMea
ns is known to have a memory leak on Windows with MKL, when there are less chunks than availa
ble threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
```



8. hét / Epilógus

Hasznos anyagok:

- Dokumentációk
  - Python hivatalos dokumentációja: https://docs.python.org/3/
  - PEP 8 Style Guide for Python Code - Melyek a jó és rossz programozási praktikák
  - NumPy hivatalos dokumentációja: https://numpy.org/doc/1.25/
- Tankönyvek
  - Dive Into Python 3
  - Dive into Deep Learning - Interaktív tankönyv Deep Learninghez
  - Fluent Python: Clear, Concise, and Effective Programming by Luciano Ramalho - Haladóbb szemléletű Python programozás
- Útmutatók
  - The Official Python Tutorial - Self-explanatory?
  - Foglalt Keyword lista - Ezeket ne használd változónévnek!

    - Codecademy - Interaktív (fizetős) online tutorial
    - CheckIO - Tanulj Pythont játékfejlesztésen kersztül
- Competitive Programming
    - Codewars
    - CodeForces

Elérhetőség

Bármilyen kérdés, kérés vagy probléma esetén keressetek minket az alábbi elérhetőségeken:

- Monori Bence - m.bence02@outlook.hu
- Wenesz Dominik - weneszdominik@gmail.com

Illetve anonim üzenetküldésre is lehetőséget biztosítunk, ezt az alábbi linken tudjátok elérni: https://forms.gle/6VtGvhja3gq6CTT66