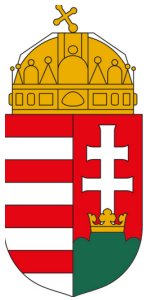
Kezdő Python programozás

SZTE Eötvös Loránd Szakkollégium



Kulturális és Innovációs Minisztérium

Nemzeti Tehetség Program

Csonka Valentin Viktor

- 3. éves programtervező informatikus hallgató
- Félév szakmai munkatapasztalat
- 2 félév oktatás az egyetemen
- Github: https://github.com/Valesz

Miért Python?

- Modern
- Könnyen használható
- A legnépszerűbb nyelv
- Kezdő barát
- Szinte akármit meg lehet vele valósítani az nagy ökoszisztémája miatt

when starting to build a new software system. The definition of the TIOBE index can be found here.

Apr 2025	Apr 2024	Change	Programming Language	Ratings	Change
1	1		Python	23.08%	+6.67%
2	3	▲	C++	10.33%	+0.56%
3	2	▼	C	9.94%	-0.27%
4	4		Java	9.63%	+0.69%
5	5		C#	4.39%	-2.37%
6	6		JavaScript	3.71%	+0.82%
7	7		Go	3.02%	+1.17%
8	8		Visual Basic	2.94%	+1.24%

Kurzus célja

- Bevezetni a programozás világába
- Eszközöket mutatni, amivel saját programozásbéli problémáinkat meg tudjuk oldani
- Alapvető programozási tudás megszerzése
- Ismerkedés haladóbb eszközökkel mint a Numpy, és Pandas

A kurzus végi tudás

- Fejlesztőkörnyezet használati tudása
- Alapvető programozási elemek ismerete
- Minimális algoritmikus gondolkodás kiépítése
- Önálló programozási problémák megoldása
- Ismeretek Python csomagokkal való munkával
- Numerikus számítások Python segítségével
- Ábrák készítése Python segítségével
- Nagy mennyiségű adatok feldolgozása Python segítségével

Feladattár

- A gyakorlófeladatokat, és nehezebb feladatokat itt találtok:
- https://drive.google.com/drive/folders/1SFGnwbCOiYqFtq-76QU_mOWX7pmRXtQZ?usp=sharing

1. Blokk: Bevezetés, első program

- Fejlesztői környezet megismerése, használata
- Első program megírása, és futtatása
- Különféle kiíratások használata
- Kommentek megismerése

Mi a Programozás?

- Receptírás, vagy útmutatás egy „vak”, de nagyon gyors és precíz asszisztensnek.
- Utasítás sorozatok adása, hogy a „vak” számítógép elvégezze a munkánkat.
- Problémamegoldás eszköze. Hogyan tudunk egy komplex problémát kisebb kezelhetőbb dolgokra lebontani.
- Automatizálás. A unalmas ismétlődő feladatok gépiesítése.

Programok futása

- A programjaink fentről lefele olvasódnak, és futnak le soronként
- Ezt nevezzük Szekvenciális programozásnak
  - Szekvenciális: Fancy (menő) szó arra, hogy egymás utáni

Programozáshoz használt eszközök

- IDE – Integrated Development Environment
  - Magyarul: Fejlesztőkörnyezet
- Fordító
  - Lefordítja a programot a gép nyelvére
  - Pythonnál erre nincs szükség
- Futtató
  - A nem lefordított programokat futtató környezet
  - Ez felel azért hogy tudjon futni a program, ha nem kellett fordító

IDE – Integrated Development Environment

(Fejlesztői környezet)

- Alapvetően egy Jegyzettömbben, és Word-ben is lehet programozni
- Az IDE-k viszont egy programozásra specializált szövegszerkesztők extra funkciókkal
  - Kód színezése
  - Kód ellenőrzése
  - Hibák keresése
  - (Kód futtatása)

Pycharm mint IDE

- A Pycharm Community egy ingyenes Pythonra specializált IDE
- Egyetemen is általában ezt használjuk
- Extra funkciói:
  - Szöveg színezése
  - Hibák észlelése
  - Szöveg kiegészítés, és javaslatok mutatása
  - Programok futtatása
  - És még sok más…

Demo – Pycharm alapvető dolgok

- Projekt létrehozása
- Projekt megnyitása
- .py fájlok készítése
- .py fájlok futtatása
- Leírt IDE funkciók megmutatása

Adatok típusa

- Szám
  - Lehet egész vagy tizedes is
  - Tizedesnél kerekítési hibákra figyelni kell
- Szöveg
  - Idézőjelek közti („ vagy , vagy `) karakterek sorozata
- Logikai
  - A megadott logikai kifejezés elvégése
  - Értéke lehet True vagy False (Igaz vagy Hamis)
  - Tagadásra a „not” Karaktert használjuk
  - Műveletek: ==, !=, <, >, <=, >=

Print függvény

- Ezzel tudunk kiírni a konzolra szöveget.
- Kódban:
  - print(„szöveg”) vagy print(számítás)

Demo – Első program

- Írjunk egy programot, ami bemutatkozik: kiírja a nevünket, az aktuális évet, és egy egyszerű számítás eredményét.

Gyakorló feladatok – Szekvenciális programozás

- Egyszerűbb feladatok:
  - https://docs.google.com/document/d/1sGjW0EnvjGrSu4mAiPiAyJOq1d7CQwNJvVG4xiPdG8o/edit?usp=sharing

- Nehezebb feladatok:
  - https://docs.google.com/document/d/1EAwhSsHNm8NkTSmtZM7xtedQrqNwTh2wC1O2SkjFXPY/edit?usp=sharing

2. Blokk: Változók és Alapműveletek

- Változók szerepének megértése
- Változók létrehozása, értékadása, módosítása
- Változóelnevezési szokások ismertetése
- Alapvető aritmetikai operátorok használata
  - Aritmetikai: Fancy (menő) szó a matematikai-ra
  - Operátorok: Műveletjelek (pl.: +, -, *, /)
- Felhasználói adat bekérése
- Típuskonverzió
- String (szöveg) formázás alapjai f-string használatával

Mi a változó?

- Nevesített tárolóhely adatok számára.
  - Pl.: költözéskor a tányéros doboz, amiben 1 lapos tányér van
- Szintaxisa (leírási módja):
  - valtozo_nev = ertek
- Az eszközünk (laptop, vagy asztali gép) memóriájában tárolódik
- Pythonban nincs a változóknak fix típusaik
  - De mindenhol máshol általában van

Változók típusai

- Integer – int
  - Egész számok (pl.: 5, 10, 1, 0, -5)
- Float – float
  - Tizedes vagy lebegő pontos számok (pl.: 0.1, 0.2, 3.1415, -2.56)
- String – str
  - Szöveg (karakterlánc)
  - Idézőjeleket használunk jelölésükre (pl.: „szöveg”, ‚locsolás’, ‚Ursula’)
- Bool – bool
  - Logikai érték (True vagy False), elágazásoknál lesz fontos
- type(változó) függvény
  - Megmondja a változó típusát

Változók elnevezése

- Beszédes nevek
  - Nem ‚a’, meg ‚b’, mert azt 1 hét múlva nem tudod mit jelent
- Elnevezési konvenciónak a „snake_case”-t használjuk
  - Minden kisbetű, és a szavakat aláhúzással választjuk el („_”)
- Nem kezdődhet számmal, vagy már lefoglalt kifejezéssel
  - Lefoglalt kifejezésekre példa: if, else, elif, print, return, def

Aritmetikai operátorok

(matematikai műveletek)

- Összeadás („+” karakterrel, pl.: 1 + 2, vagy a + b)
- Kivonás („-” karakterrel, pl.: 1 – 2, vagy a - b)
- Szorzás („*” karakterrel, pl.: 1 * 2, vagy a * b)
- Hagyományos osztás („/” karakterrel, pl.: 1 / 2, vagy a / b)
- Egész osztás („//” karakterekkel, pl.: 3 // 2, vagy a // b)
  - A tizedes részt levágja, nem kerekít
- Maradékos osztás („%” karakterrel, pl.: 1 % 2, vagy a % b)
  - Visszaadja a maradékot
- Műveleti sorrend változtatása (és) karakterekkel

Felhasználói bemenet

- input(„kérdés”) bekér adatot a felhasználótól
- Fontos, hogy mindig String (szöveg, str) típust kapunk még ha szám is amit megadott a felhasználó
- El lehet tárolni változóban
  - Pl.: valtozo_nev = input(„kérdés”)

Típus konverzió (típus váltás)

- Lehet a típusok között váltani, csak figyeljünk rá, hogy mit mire váltunk
  - Például egy szöveget nem tudunk számmá alakítani, csak ha maga a szöveg is egy szám
- tipus(átváltandó dolog)
  - Pl.: int(„3”)
  - Itt a 3 mint szöveg szerepel, és ezt váltjuk át egy egész számmá
- Ha nem jól végezzük az átváltást a programunk hibát dob

Demo - Változók

- Kérjük be a felhasználótól a nevét és születési évét. Számoljuk ki az életkorát és üdvözöljük név szeirnt, kiírva az életkorát is.

Gyakorló feladatok - Változók

- Egyszerűbb feladatok:
  - https://docs.google.com/document/d/1wTNCUEbgP1gWUFrX_UMinb20SKa6NJBvOPst-_po6uE/edit?usp=sharing

- Nehezebb feladatok:
  - https://docs.google.com/document/d/1FTv6MP43SrHnHnuslgBsGCzsy6q5rDgZCdbE9pq8R6w/edit?usp=sharing

3. Blokk: Listák

- Listák szerepének megértése
- Listák létrehozása
- Elemek elérése elölről, és hátulról is
- Listák szeletelése
- Listaelemek módosítása
- Alapvető lista függvények ismerete
  - append, insert, remove, pop, len, sort, reverse

Mi a lista?

- Elemek gyűjteménye
- Módosítható elemek
- [és] karakterek jelölik (szögletes zárójel)
- Akármelyik elemet el tudjuk érni a listából
  - Pl.: lista_neve[elem indexe]
- Elemeit vesszővel elválasztva fel tudjuk sorolni
- Különböző típusú elemeket is tárolhat, de általában azonos típusú elemeket szokott tárolni
- Dobozba belerakjuk a tányérjainkat például

Listák indexelése (lista elemeinek sorszáma)

- Fontos, hogy a listák 0-tól kezdődnek!!!!
  - Tehát az 1. elemre, a 0 index-el tudunk hivatkozni
- lista_neve[0] az első elem
- lista_neve[tetszőleges index] a tetszőleges elem elérése
- Tudunk negatív indexeket is használni, ekkor hátulról számolja
  - lista_neve[-1] az utolsó elem, és lista_neve[-2] az utolsó előtti
- Figyeljünk, hogy ne hivatkozzunk nem létező elemre, mert akkor is hibát dob a programunk

Listák szeletelése

- Listát szét tudunk vágni a következő módon:
  - lista_neve[kezdő index:végső index:lépésköz]
  - Pl.: list[1;-1;2] ez a lista minden második elemét veszi ki, kivéve az utolsót
- Ezeket akármilyen sorrendben lehet alkalmazni
  - Pl.: list[4:] kiveszi a negyedik elem után az összeset
  - Pl.: list[:4] kiveszi az első 4 elemet
  - Pl.: list[2:4] kiveszi a 2. elemtől a 4-ig
  - Pl.: list[4::2] kiveszi a negyedik elemtől minden másodikat
  - Pl.: list[:4:2] kiveszi a negyedik elemig minden másodikat
- Negatív lépésköz is lehet
  - Pl.: list[::-1] megfordítja a listát

Listák módosíthatósága

- Listák egyes elmeit tudjuk módosítani
- lista_neve[elem indexe] = új érték
  - Pl.: list[2] = „vidám” megváltoztatja a lista 2. elemét a „vidám” szövegre
  - Pl.: list[-1] = 3 megváltoztatja a lista utolsó elemét a 3 számra

Alapvető lista függvények

- lista.append(elem) beszúrja a lista végére az elemet
- lista.insert(index, elem) beszúrja a lista adott indexére az elemet
- lista.remove(elem) kiveszi a listából az elemet
  - Ha nincs benne az elem, akkor hibát dob
- lista.pop(index) kiveszi az adott indexről az elemet
  - És vissza is adja, szóval el lehet tárolni egy változóban
- len(lista) lista elemeinek számát adja vissza (length rövidítése)
- lista.sort() rendezi a listát növekvő sorrendben
- lista.reverse() megfordítja a lista elemeinek sorrendjét

Demo - Listák

- Készítsünk egy egyszerű bevásárlólista-kezelő programot. Lehessen hozzáadni tételt, megtekinteni a listát, és eltávolítani tételt.

Gyakorló feladatok - Listák

- Egyszerűbb:
  - https://docs.google.com/document/d/1NRCgycLlYuwxGoktZ0kk9ijRnRbRBJpbQLBi3STHcK0/edit?usp=sharing

- Nehezebb:
  - https://docs.google.com/document/d/1iyS665TWN3JSwREiZyeb3PT5sMOo_c6fiPqHFdljgpw/edit?usp=sharing

4. Blokk: Elágazások

- Egyes részek futását feltételhez kötni
- Ismerni az if, elif, else kulcsszavakat, és szerkezetüket
- Megérteni az indentáció (behúzás) kritikus szerepét a Pythonban
- Összehasonlító operátorok használata
- Logikai operátorok használata
- Egyszerű és összetett feltételek megfogalmazása

Indentáció (behúzás)

- Jelöli és egybefogja a kódrészeket
- Pythonban kötelező szintaktikai elem

Mi az elágazás?

- Lehetővé teszi a program különböző módon való futását feltételek teljesülésekor
- Útelágazás a programunkban

Az if utasítás

- Alapvető feltételes szerkezet
- if feltétel:
  - A feltétel után : karakter kell
  - Utána a következő sor behúzott sor(ok) csak akkor hajtódnak végre, ha a feltétel igaz (True) volt

Az elif utasítás

- Az „else if” rövidítése
- Lehetővé teszi több feltétel egymás utáni láncolását
- Akkor értékelődik ki, ha az előző if (vagy elif) feltétele hamis volt
- Szintaxis:
  - elif feltétel:
  - Behúzással jelöljük a hozzá tartozó kód blokkot
- Több elif is lehet egymás után, de mindig egy if után lehet csak

Az else utasítás

- Akkor hajtódik végre, ha az if és az összes elif feltétele hamisra (False) értékelődött ki
- Szintaxis:
  - else:
  - Majd behúzással jelöljük a hozzá tartozó sorokat
- Egy if-hez maximum 1 else tartozhat
- Az else blokk mindig a if és elif utasítások után kerül

Összehasonlító operátorok

- Egyenlőség: ==
- Nem egyenlő: !=
- Nagyobb: >
- Kisebb: <
- Nagyobb egyenlő: >=
- Kisebb egyenlő: <=
- Eredményük mindig logikai (True, vagy False)

Logikai operátorok

- feltétel1 and feltétel2
  - Igaz ha mind a két oldalán szereplő feltétel igaz
- feltétel1 or feltétel 2
  - Igaz ha a 2 feltétel közül valamelyik igazra értékelődik ki
- not feltétel1
  - Megfordítja a feltétel1 értékét
  - Tehát ha igaz (True) volt akkor hamis (False) lesz, és fordítva is így van

Demo - Elágazások

- Kérjünk be egy vizsgapontszámot (0-100). Írjuk ki a szöveges értékelést: 0-49: "Elegtelen", 50-69: "Elégséges", 70-84: "Közepes", 85-94: "Jó", 95-100: "Kiváló". Kezeljük az érvénytelen (0 alatti vagy 100 feletti) bevitelt is.

Gyakorló feladatok - Elágazások

- Egyszerűbb:
  - https://docs.google.com/document/d/1agdPzxPKGceP-swl14zY_uYJXxrQsCC77cwa2L4weo0/edit?usp=sharing

- Nehezebb:
  - https://docs.google.com/document/d/14oWHzLhvvfKy_Wi0VBbanu9i2gtSGrM2bE8bbkF3NaI/edit?usp=sharing

5. Blokk: Ciklusok

- Megérteni miért van szükség a ciklusokra, és miért hasznosak
- Ismerni a for ciklust és használatát sorozatok (listák, szövegek, range) bejárására
- Ismerni a while ciklust, és használatát feltételtől függő ismétlésre
- Megérteni a break és continue utasításokat
- Tudni egyszerű ciklusokat írni adatfeldolgozásra, ismétlésre

Miért kellenek a ciklusok?

- Feladatok (kódblokkok) többszöri végrehajtása, anélkül hogy többször leírnánk azt
- Adatsorozatok (listák, szövegek) elemeinek feldolgozására

A for ciklus

- Iteráció (végiglépkedés) egy bejárható sorozat elemein
- Szintaxis:
  - for ciklus_valtozo in sorozat:
  - A ciklus_valtozo minden iterációban felveszi a sorozat következő elemét
- Listákra:
  - for ciklus_valtozo in lista_neve:
- Szövegre:
  - for ciklus_valtozo in szoveg:
- Range használata következő dián

A for ciklus range használatával

- A range utasítás egy számsorozatot generál
- Általában for ciklushoz szoktuk használni
  - for ciklus_valtozo in range(start, stop, step):
- Szintaxis:
  - range(stop) 0-tól stop -1-ig (pl.: range(5) -> [0,1,2,3,4]
  - range(start, stop) start-tól stop -1-ig (pl.: range(2, 6) -> [2,3,4,5]
  - range(start, stop, step) start-tól, stop-ig step lépésközönként

A while ciklus

- Addig ismétli a kód blokkot ameddig a feltétel igaz (True)
- Szintaxis:
  - while feltétel:
  - Utána behúzással a hozzá tartozó kód blokk
- Fontos! Gondoskodjunk arról, hogy a feltétel valamikor hamissá válik, mert különben egy „végtelen ciklusba” esünk, és lefagyhat a programunk
  - Ezt általában a feltételben szereplő változó növelésével, vagy csökkentésével szoktunk elérni, vagy más kilépő részek segítségével

Ciklusvezérlő utasítások

- break
  - Azonnal kilép a jelenlegi ciklusból
- continue
  - A blokk hátralévő részét átugorja, és egyből kezd egy új iterációt

Demo - Ciklusok

- Írjunk ki minden elemet egy listából for ciklussal, és számoljuk össze hány páros szám van

Gyakorlófeladatok - Ciklusok

- Egyszerűbb:
  - https://docs.google.com/document/d/1EPEkviTRXtdUHtTCQHPXNX_IZFSW2t9L8NDsToEclFE/edit?usp=sharing

- Nehezebb:
  - https://docs.google.com/document/d/1KEH4jo7TrryCUSL4oZYOROWRWvWCPIyzvJccWwR6D8U/edit?usp=sharing

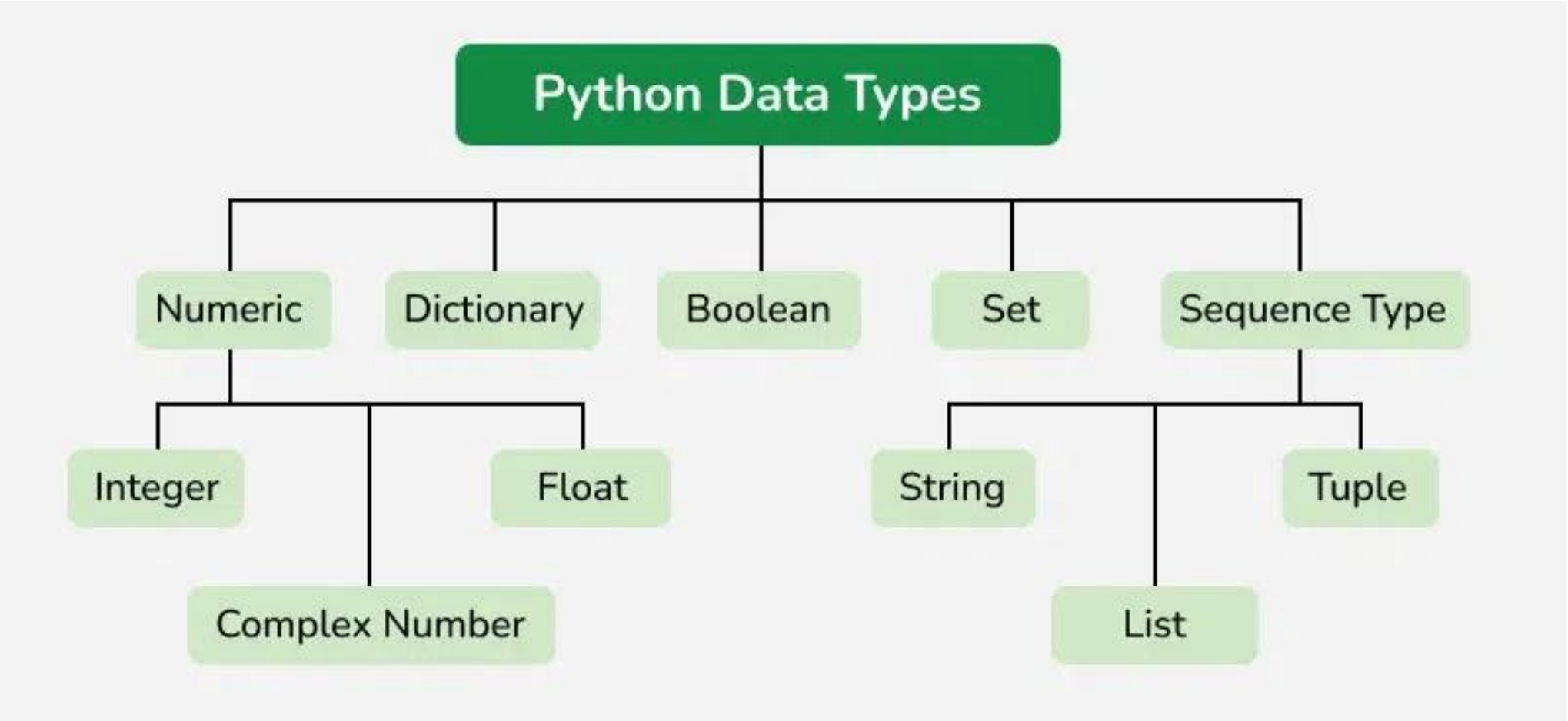
6. Blokk - Alapok Ismétlése

- Megerősíteni a programozás alapvető építőköveinek megértését.
- Növelni a magabiztosságot összetettebb, több elemet követelő problémák megoldásában.
- Belelendüljünk a programozásba újra.

Szekvencia

- A programok fentről lefelé futnak
  - A változónk csak onnantól lefelé lehet használni, hogy deklaráltuk (létrehoztuk) azt
- print függvényt használjuk kiíratásra
  - Több féle módon tudunk külön típusú változókat összekötni

Változók


Python Data Types
Numeric
Dictionary
Boolean
Set
Sequence Type
Integer
Float
Complex Number
String
Tuple
List

Egyszerű Változók

- Számok
  - Integer – Egész szám
  - Float – Tizedes szám
- Szöveg
  - String – szöveg, speciális lista ami karakterekből áll
- Logikai
  - Állítások amik igazak (True) vagy hamisak (False) lehetnek
  - and, or, not kulcsszavak és logikai operátorok (logiakai jelek, pl.: egyenlő)

Összetett Változók

- Lista
  - Egyszerű változók „halmaza”
  - Általában ugyanolyan típusú változókat rakunk bele
  - Lehet belerakni, elvenni, és módosítani elemet
- Dictionary (Szótár)
  - Egy Lista ahol minden elemnek van egy címkéje
  - Felfogható elemek megfeleltetésének
  - Például: Magyar Értelmező Kéziszótár, ahol egy szóhoz kapcsolódik a jelentése, és több szó van egy példányban

Típus konverzió (Típus váltás)

- Nem mindegy hogy milyen típusban használunk egy változót
- Lehet hogy át kell váltanunk a típusát
- Példa: az input() függvény mindig szöveget ad vissza (még ha számot is írtunk be neki)

- Szintaxis: tipus(változó)
  - Példa: int(„23”)
  - A példa egy szöveget (ami a 23 szöveg) alakítja át egy egész számmá, amivel már tudunk műveleteket csinálni

Felhasználói bemenet

- Input függvényt használjuk ennek a kérésére
- Az input függvény mindig egy szöveget ad nekünk
- Legyünk jó programozók és csináljunk felhasználóbarát alkalmazást, feltehetünk kérdéseket amire várjuk a választ

- Szintaxis:
  - Példa: input(„Van-e kérdésed? ”)
  - A példa kiírja a konzolra, hogy „Van-e kérdésed? „, majd vár a felhasználótól egy szöveget

Listák

- Egyszerű változók „halmaza”
- Általában ugyanolyan típusú változókat rakunk bele
- Lehet belerakni, elvenni, és módosítani elemet
- Le tudjuk kérni a lista méretét (hány elem van benne)
- Itt már keletkezhetnek hibák is a programunkban
  - Ha olyan elemet szeretnénk elérni ami nincs a listában (pl.: egy 2 elemű listában a 10-et szeretnénk visszakérni)
  - Vagy ha nem létező elemet szeretnénk a remove-al kitörölni

Alapvető lista függvények

- lista.append(elem) beszúrja a lista végére az elemet
- lista.insert(index, elem) beszúrja a lista adott indexére az elemet
- lista.remove(elem) kiveszi a listából az elemet
  - Ha nincs benne az elem, akkor hibát dob
- lista.pop(index) kiveszi az adott indexről az elemet
  - És vissza is adja, szóval el lehet tárolni egy változóban
- len(lista) lista elemeinek számát adja vissza (length rövidítése)
- lista.sort() rendezi a listát növekvő sorrendben
- lista.reverse() megfordítja a lista elemeinek sorrendjét

Elágazások

- Ha ugyanolyan mértékű behúzást használunk egymás utáni sorokon akkor azokat egy „blokkba” tudjuk foglalni
- If/elif/else segítségével tudunk elágazásokat létrehozni
- Úgy kell elképzelni mintha egy elágazáshoz érnénk, és valamilyen logika alapján elkezdjük balról jobbra nézni az utakat és aminél először lesz jó a logikánk ott indulunk el

Logikai operátorok (logikai műveletek)

- Nagyon összetett logikákat tudunk létrehozni a helyes használatukkal
- and, or, not, <, >, <=, >=, ==, !=, (,) műveleteket használhatunk
- Mindig vagy igazat (True), vagy hamisat (False) eredményeznek

Ciklusok

- For ciklus
  - Listán, vagy range-en (számsoron), vagy szövegen való végig lépegetés
- While ciklus
  - Addig csinálunk valamit ameddig Hamis nem lesz a logikánk
  - Végtelen ciklust eredményezhet (Ez veszélyes, mert lefagyhat a futtató felületünk)
- Break utasítás
  - Kilép a legbelső ciklusból, és az utána lévő utasításokat már nem hajtja végre
- Continue utasítás
  - Átlép a következő iterációra, és az utána lévő utasításokat már nem hajtja végre

Demo - Ismétlés

- Írjunk egy egyszerű programot, ami bekér 5 db vizsgaeredményt (0-100) a felhasználótól, tárolja őket egy listában. Majd számolja ki az átlagpontszámot, és írja ki az átlagot, valamint azokat az eredményeket, amik az átlag felett vannak. Kezeljük a hibás bevitelt (nem szám, vagy tartományon kívüli érték).

Gyakorlás - Ismétlés

- Egyszerűbb feladatok:
  - https://docs.google.com/document/d/1DpFqrEV37p8d_sBkuES4n-E2atUixxV3wTqYhozgTXQ/edit?usp=drive_link

7. Blokk - Függvények

- Megérteni a függvények célját
- Tudni függvényeket definiálni, és hívni
- Megérteni a paraméterek szerepét
- Tudni visszaadni értéket a függvényből
- Alapvető ismeretek a változók hatásköréről

Miért van szükségünk függvényekre?

- DRY – Don’t repeat yourself
  - Kódismétlődést elkerüljük. Ha egy feladatot többször el kell végeznünk, akkor írjunk rá függvényt
- Olvashatóbb és struktúráltabb legyen a kódunk
  - Nagyobb egybefüggő kódrészek logikai egységekre bontása. A függvény neve leírja mit csinál a blokk, ezért nem kell értelmeznünk a kódot.
- Újrafelhasználhatóság
  - Máshol is fel tudjuk használni a kódunkat.

Függvények definiálása

- Szintaxis:
  - def függvény_neve(paraméter1, paraméter2, ...):
- A függvény neve fontos hogy beszédes legyen
- Itt is snake_case-t használunk elnevezéskor
  - Kisbetűk, szavanként aláhúzással elválasztva
- Paraméterek azok változók amiken keresztül adatot tudunk átadni a függvényünknek

A return utasítás

- Értéket ad vissza a függvény hívásának helyére
- Hatására megszakad a függvény futása, és a további részek már nem kerülnek lefutásra, viszont a futást a függvény hívásának helyéről folytatja
- Ha nincs return utasítás akkor a függvény egy None értéket ad vissza
- Egy függvényben lehet több return utasítás is
  - Pl.: egy if/else ágban

A változók hatásköre

- Lehet lokális és globális

A változók hatásköre - Lokális

- A függvényen belül létrehozott változók
- Csak a függvényen belül érhetőek el
- A függvény befejezésekor megszűnnek

A változók hatásköre - Globális

- A függvényen kívül definiált változók
- Bárhonnan el lehet érni őket olvasásra
- Jobb paraméterként átadni, és return-el visszaadni valamilyen értéket

Demo - Függvények

- Írjunk egy funkciót, ami kiszámolja egy téglalap területét és kerületét, és egy másikat, ami üdvözöl egy felhasználót.

Gyakorlófeladatok - Függvények

- Egyszerűbb:
  - https://docs.google.com/document/d/1GlhNswi-99NgKt4ujqMG0aGbCjFWqUBQ1aWcSk8CXqc/edit?usp=sharing

- Nehezebb:
  - https://docs.google.com/document/d/1IaH1xp9S2fGGtTYVg--SsDZv856iBvgjFZMEPAakjW0/edit?usp=sharing

8. Blokk – Külső csomagok

- Megérteni a külső csomagok fogalmát, és hasznosságát
- Ismerni a PyPi-t (Python Package Index) mint központi tárolót
- Megtanulni a pip csomagkezelő alapvető parancsait
- Megérteni a virtuális környezetek (venv) fontosságát
- Tudni telepíteni és importálni külső csomagokat Python szkriptben

A Python ereje, a csomagok!

- A Python alapvetően sok mindent meg tud csinálni
- Az igazi ereje viszont a hatalmas közösség által fejlesztett csomagokban rejlik.
- A csomagok előre megírt kódgyűjtemények speciális feladatokra
  - Pl.: Adatfeldolgozás, webfejlesztés, képfeldolgozás, játékok, stb…
- Példa: Tűzgyújtás eszközökkel, és eszközök nélkül

PyPi (Python Package Index)

- https://pypi.org/
- A hivatalos központi hely, ahonnan a legtöbb publikus Python csomag letölthető

pip – Python csomagkezelő

- A python beépített csomagkezelője
- Ezzel tudunk csomagokat letölteni, és letörölni
- Általában parancssorból szoktuk használni, de Pycharmba van beépített csomagkezelő pip használatával

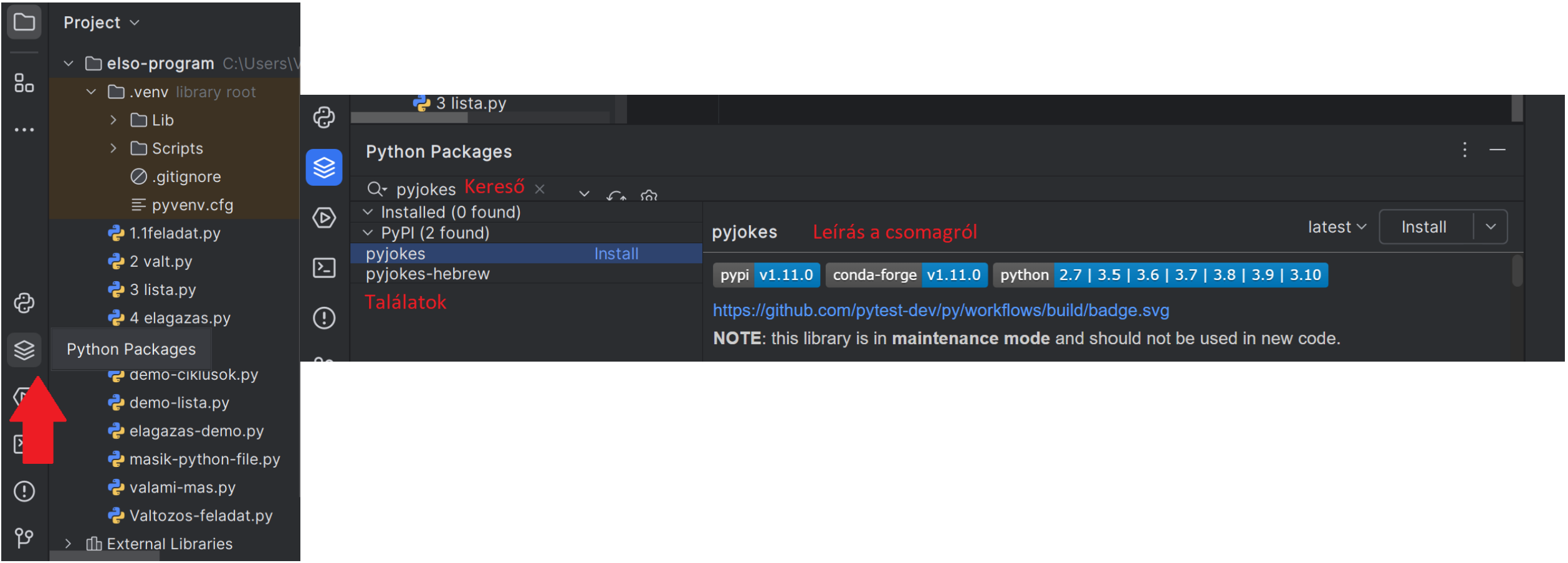
pip parancsok – I

- pip install csomag – Letölti a „csomag” nevű csomagot
- pip install csomag==verzio – Letölti a „csomag” csomagot a „verzio” verzióval
- pip list – Kilistázza a telepített csomagokat
- pip show csomag – Részleteket ír ki a csomag nevű csomagról
- pip uninstall csomag – Letörli a telepített csomag nevű csomagot

pip parancsok – II

- pip freeze – Kilistázza a csomagokat úgy, hogy azokat 1 paranccsal lehessen telepíteni.
- pip freeze > requirements.txt – Belerakja a „requirements.txt” fájlba a csomagok leírását, úgy hogy azt 1 paranccsal lehet telepíteni
- pip install –r requrements.txt – Telepíti a requirements.txt-ben található csomagokat

Pycharmba épített csomagkezelő



Demo - Csomagok

- Hozzunk létre egy új projektet PyCharm-ban (figyelve, hogy létrejön-e a venv). Telepítsük a pyjokes és emoji csomagokat. Írjunk egy rövid szkriptet, ami kiír egy véletlen viccet a pyjokes segítségével, majd kiír egy emojit az emoji csomag használatával. Generáljunk requirements.txt-t.

Gyakorló - Csomagok

- Egyszerűbb feladatok:
  - https://docs.google.com/document/d/1tc94zUmgp0DMX4uG-crQ7FcXeJTGRmeHJd1CniYezMc/edit?usp=drive_link

9. Blokk – Pandas csomag

- Megismerkedni a Pandas könyvtárral mint alapvető adatelemzési eszközzel.
- Megérteni a két fő Pandas adatszerkezetet: Series és DataFrame.
- Tudni létrehozni egyszerű Series-t és DataFrame-et Python listákból/szótárakból.
- Képesnek lenni alapvető adatelérési és -manipulációs műveletekre: oszlopok/sorok kiválasztása, adatok megtekintése
- Tudni beolvasni adatokat egyszerű CSV fájlból.

Mi a Pandas?

- Magas szintű Python könyvtár gyors, rugalmas és kifejező adatszerkezetekkel, kifejezetten táblázatos és idősoros adatok kezelésére tervezve
- Az adatelemzés svájci bicskája Pythonban
- Főbb felhasználási területek: adatbeolvasás/-írás (CSV, Excel, SQL stb.), adattisztítás, adattranszformáció, elemzés, vizualizáció előkészítése

Alap adatszerkezetei - Létrehozás

- Series létrehozása
  - pd.Series(adat) – Az adat lehet egy lista, vagy dictionary
  - sr-nek szoktuk rövidíteni változóknál

- DataFrame létrehozása
  - pd.DataFrame(data, columns=[oszlopok]) – Az adat lehet egy lista, vagy dictionary ahol a kulcsok (címkék) az oszlopnevek, és az értékek a listák/Series-ek
  - df-nek szoktuk rövidíteni változóknál

Alapvető műveletek – I (Kiíratások)

- df.head(n) - Kiírja az első „n” sort
- df.tail(n) - Kiírja az utolsó „n” sort
- df.info() - Kiír adatokat a DataFrame-ről
- df.describe() – Statisztikai összefoglalót ad a DataFrame-ről
- df.shape() – Sorok, és oszlopok számát írja ki

Alapvető műveletek – II (Oszlopok)

- df[„oszlopnév”] vagy df.oszlopnév – egy Series-t ad vissza, ami az adott oszlophoz tartozik
- df[[„oszlop1”, „oszlop2”]] – Több oszlopot választ ki, és egy DataFrame-et ad vissza

Alapvető műveletek – III (Sor)

- df[0:3] – Szeleteléssel úgy ahogyan listáknál is
- df.loc[„index-címke”] – A címke alapján
- df.iloc[0] vagy df.iloc[0:3] – A megadott pozíción lévő oszlop
- df[feltétel] – Kiválasztja azokat a sorokat ahol a feltétel igaz
  - df[df[„oszlop”] > ertek] – Használata
  - Pl.: df[df[„suly”] > 70] – Kiválasztja a sorokat ahol a súly nagyobb mint 70

Alapvető műveletek – IV (Adatbeolvasás)

- pd.read_csv(„fajlnev.csv) – Csv adattípus beolvasására
- pd.read_excel(„fajlnev.xlsx”) – Excel beolvasására

Demo - Pandas

- Hozzunk létre egy Pandas DataFrame-et néhány hallgató adatával (név, kor, szak, átlag). Végezzünk el néhány alapvető műveletet: jelenítsük meg az első pár sort, az oszlopinformációkat, válasszuk ki csak a neveket, majd szűrjük ki azokat a hallgatókat, akiknek az átlaga egy bizonyos érték felett van.

Demo - Pandas

- Hozzunk létre egy Pandas DataFrame-et néhány hallgató adatával (név, kor, szak, átlag). Végezzünk el néhány alapvető műveletet: jelenítsük meg az első pár sort, az oszlopinformációkat, válasszuk ki csak a neveket, majd szűrjük ki azokat a hallgatókat, akiknek az átlaga egy bizonyos érték felett van.

Gyakorlás - Pandas

- Egyszerűbb feladatok:
  - https://docs.google.com/document/d/1n_KvY8G4OQ_uYxVVOHWmj_PZ8O-bu5gccjbksPzQbMU/edit?usp=drive_link