Mentor Klub

Python tanfolyam

9. alkalom

Gyakorló- és házi feladatok - megoldás

- Készítsünk programot, egy input.txt fájlban lévő szavakat kezdőbetű szerint megszámol, és az eredményt egy dict()-ben kiírja a standard outputra.
- Fejlesszük tovább a programot, hogy a végeredményt egy output.json nevű fájlba írja ki, JSON formátumban
- Fejlesszük tovább a programot, hogy a kimenet ne csak a darabszámot tartalmazza, hanem a szavakat is.
  - A kimeneti formátum a következő legyen
    - {"a":{"darab":3, "szavak": ["alma", "aki", "az"]} , "z": {"darab":1, "szavak": ["zászló"]}}
- Fejlesszük tovább úgy a programot, hogy a kimeneti fájlban a szavak ábécé sorrendben legyenek a szavak tömbben.

Hibakezelés – Miért fontos?

Mentor Klub

- Bármilyen szoftver forráskódjának kb 30-40%-át teszi ki az üzleti logika (*business logic)*, azaz az, ami a szofver valódi haszna.
- A többi rész (igen, 10000 sor esetén 6000 sor) nem más, mint hibakezelés, input validálás.
- Azaz olyan vezérlés megvalósítása, mely minimalizálni tudja (eliminálni úgysem lehet) az olyan lefutásokat, mely nem megengedett állapotba hozná a programot.
    - Nem megengedett vagy inkonzisztens adat az adatbázisban
    - Végzetes hiba, azaz váratlan szoftverleállás
- Egy jól működő szoftverfejlesztési projekten mindent meg kell tenni, hogy bullet-proof legyen az elkészült program.
    - Sajnos nem mind jól működő…

Hibakezelés – Teljes program szintjén

- Egy program futásának eredménye két féle lehet: Helyes vagy hibás.
- A Helyes lefutást sok esetben 0-s hibakódú futásnak hivják.
  - Ezért van nagyon sok programozási nyelvben exit(0) vagy return 0 a programkód végén.
  - Linux operációs rendszerekben ezt a viselkedést ma is nagyon sokszor kihasználják.
- Azt lehet mondani, hogy bármilyen 0-tól különböző exit code hibát jelent.
  - A programot inditó script meg tudja vizsgálni a futtatott parancs kilépési kódját, és az alapján folytatódik a vezérlés.
  - Például Windows batch fájlok (*.bat) , vagy Linux bash scriptek(*.sh).

Hibakezelés a programon belül

- Természetesen ha a programunk hibás kilépési kóddal *terminál* (fejeződik be), akkor a programon belül valamilyen vezérlés fogja ezt a hibakódot adni.
- Gyakori megközelités, hogy ha hiba történik, akkor kiírjuk a hibát a képernyőre. **Igen ám, de…**
- Mivel a programunk struktúrált, azaz függvényekből (alprogramokból áll), ezek a függvények egymást hívják.
- Viszont nem igazán tudják olvasni a monitorra kiírt szöveget, és azt értelmezni. (Még nem… lásd chatGPT ☺)

Mire épitsünk?

- Az 1990-es évekig az a megközelités élt, hogy a függvényeknek legyen visszatérési értékük. Ha ez 0, akkor sikeres volt a lefutás, ha attól különböző, akkor sikertelen.
- Az, hogy az egyes értékek mit jelentettek, a függvények dokumentációjában adták meg.
- Külön szépség *volt (?)*, hogy a C, C++ világban a 0 a hamis-t jelenti, igy a sikeres lefutás ellenőrzése igy nézett ki:
  - `if (!fuggveny_hivas()) {…}`
- A kivétel pedig erősiti a szabályt, szóval mindíg voltak olyan függvények, amik nem követték ezt az ajánlást.

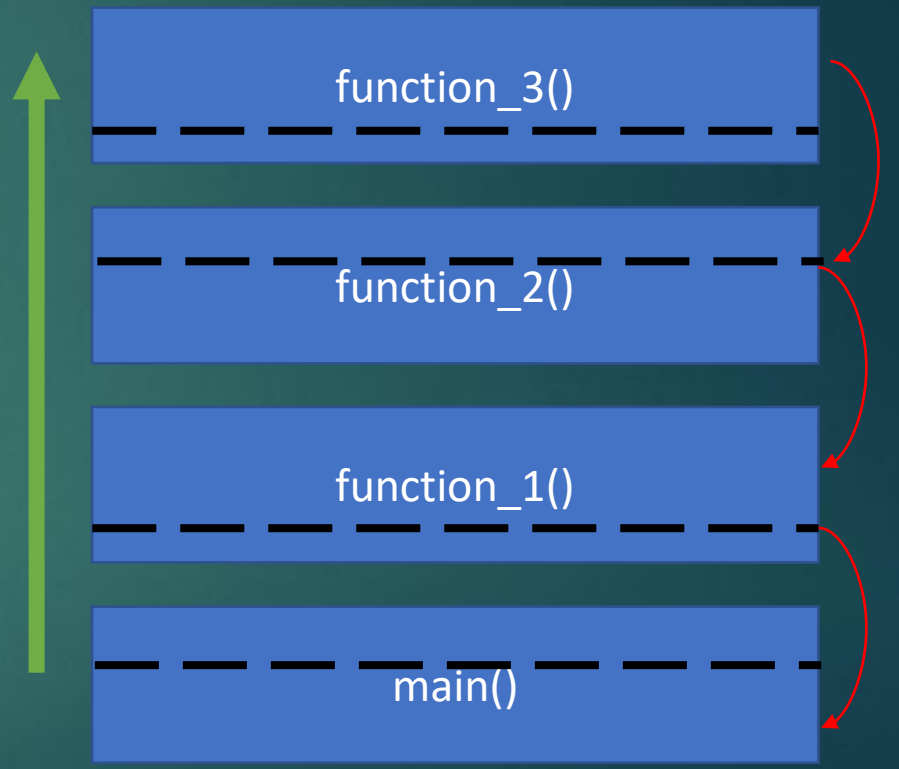
De ettől még…

Mentor Klub

- A megközelítés helyes.
- A programon belül a függvények a gép számára egyértelműen értelmezhető formában jelzik, hogy valami nincs rendben.
- Ez történhet visszatérési értéken keresztül
  - Nincs nyelvi támogatás, a programozó (csapat) dönt, hogy melyik függvénynél milyen érték számit hibának.
- Vagy történhet ún. kivétel (exception) kiváltásával
  - Azért kivétel a neve, mert a program futása során valamilyen kivételes, nem várt (dehogynem) esemény történt.
  - A kivételeket a modern programozási nyelvek egytől egyig támogatják.

Kivételek

- Amikor egy kivétel (ez egy objektum) kiváltódik, akkor az adott függvény félbeszakad, és a kivétel elkezd terjedni a hivó függvény irányába.
- Ha az adott függvény kezeli a kivételt, akkor az a függvény “lenyeli” a hibát.
- Ha nem kezeli, akkor a kivétel tovább terjed a hivó felé.
- Ha sehol sem kezelik, akkor a program hibakóddal terminál.
  - A kivétel “felbuborékozik” a felszínre.



Kivételek kezelése - szintaxis

```
try:
   utasitas1
   utasitas2
except KIVETEL_TIPUS as VALTOZO:
   hibakezelo_utasitasok1
except MASIK_KIVETEL_TIPUS as VALTOZO:
   hibakezelo_utasitasok2
finally:
   utasitas3
```

Kivételek kezelése - szemantika

- A `try` blokk tartalmát *megpróbáljuk* végrehajtani.
  - Tudatában vagyunk annak, hogy itt történhet olyan hiba, amit helyben kezelni szeretnénk. Ezért csináltuk ezt a blokkot.
- Hiba esetén a program futása megszakadna, de mivel `try` blokkban történt a hiba, ezért a keletkezett kivételt az `except` blokkokban *elkaphatjuk*.
  - Mi határozzuk meg, hogy milyen típusú kivételeket kapunk el. Nem kötelező mindegyiket.
  - Különböző típusú kivételeket különbözőképpen is kezelhetünk.
- Azokat a lépéseket amiket akár sikeres futás esetén akár hiba esetén szeretnénk végrehajtani, a `finally` blokkba kell tenni.

Kivétel kiváltása, saját kivétel készítése

Mentor Klub

- Kivételt a `raise` paranccsal válthatunk ki.

```
raise Exception("Valamilyen uzenet")
```

- Készíthetünk saját kivétel típusokat is

```
class SajatKivetel(Exception):
    pass
```

- Így már írhatjuk, hogy

```
raise SajatKivetel("JAJ Hiba!!!")
```

Gyakorló- és házi feladatok

- Készítsünk interaktív számológép programot.
- Induláskor kérjen be a felhasználótól egy egyszerű matematikai feladatot, mely 2 operandusból és 1 operátorból (+,-,*,/) áll.
  - A négy alapműveletet kell támogatni, a megfelelő matematikai szabályokkal.
    - Például a 0-val történő osztás nem engedélyezett. (Hiszen nincs értelme)
- Helyes feladat esetén a program számolja ki az eredményt.
- A programból az exit paranccsal lehet kilépni, melyet a felhasználó ír be.
- A program mindaddig kérjen be új feladatot, amíg a felhasználó nem lép ki.
- Hibás feladat esetén a program írja ki, hogy hibás feladat, és a hibás feladat okát is írja ki.
- Használjunk kivételkezelést és bontsuk függvényekre a programunkat.
- Az egyes függvények legyenek némák, azaz nem írhatnak ki semmit a kimenetre. Csak a főprogram írhat a kimenetre.

Gyakorló- és házi feladatok (példa)

```
Kérem a feladatot: 2+5
7
Kérem a feladatot: 5/0
Hibás feladat, nullával nem lehet osztani
Kérem a feladatot:6-alma
Hibás feladat, nem numerikus operandus
Kérem a feladatot 6=8
Hibás feladat, nem támogatott operátor
```