Python programozás a gyakorlatban - Kötelező program

Általános követelmények

- A programnak működő Python 3.10 programnak kell lennie.
- A beadott programnak csomagoltnak kell lennie, amit a `pip install <útvonal>` paranccsal feltelepíthetünk.
- A program működésében fontos osztályok egy részéhez unit tesztet is készíteni kell. Amennyiben másfajta tesztelés is szükséges, úgy azokat is be kell mutatni. Azonban nem kell a részletekben elveszni, 50 % tesztlefedettség elegendő.
- A program készítése közben szem előtt kell tartani az általános Python best practice-eket, kódolási stílust, pythonic kódolást, ezért beadáskor legalább 2 kódelemző eszköz eredményét is csatolni kell. Természetesen ezeket használni kell fejlesztés közben is. A kritikus hibákat illendő volna kijavítani. Az eszközök lehetnek például:
    - Pylint
    - Flake8
    - pycodestyle (pep8)
    - Prospector (ez magába foglal több másik eszközt is)
    - pyroma
    - Bandit
    - Bármi, adott domainhez illeszkedő statikus elemzőeszköz
- A feladatok elkészítése során kerüljük a beégetett értékeket, lehetőség szerint konfigurációs fájlokat használjunk.

Értékelési szempontok

- Objektumorientált szemlélet alkalmazása (logikusan legyenek osztályok használva)
    - Bármely feladatnál értelmesen használva az OOP-t, legalább 5 osztály szükséges lehet (de persze lehet, hogy sokkal több fog kelleni)
- Feladat megoldásának minősége
    - A kitűzött feladatleírásnak mennyire felel meg az alkalmazás.
- Internetről talált kódok beépítése
    - Bármit lehet használni a megoldás során, de amit az internetről másoltunk (akár egy az egyben, akár kisebb módosításokkal), azt jelölni kell. A bemásolt kódoknak bele kell épülniük a rendszerbe, nem lehetnek "fekete bárányok".
- Utánajárás
    - Az órai anyaghoz képest mennyi új ismeretet tartalmaz, mennyire mélyült el a hallgató az adott témakörben.
- Domain-specifikus eszközkészlet
    - A kitűzött feladat megoldásához mennyire használja a hallgató az adott domain legnépszerűbb függvénykönyvtárait/megoldásait.

Választható feladatok

Asztali/mobil alkalmazás

Követelmények

- Legalább 4 különböző ablaktípus használata (elrendezés/funkcionalitás)
- Többször használt komponensek kiszervezése külön osztályba (például ha több helyen van kereshető táblázat, azokat lehet érdemes kiszervezni egy saját osztályba, és a kész komponenst használni több helyen)

Hasznos függvénykönyvtárak (csak példának van összeszedve néhány, természetesen bármit lehet használni)

- Tkinter
- PyQt
- PyGObject
- Kivy (mobil alkalmazás esetén)
- **Ábrákhoz**: Matplotlib, Plotly, Seaborn

Feladat ötletek

- **Előfizetéseket kezelő alkalmazás**: A felhasználó asztali kliensen keresztül felveheti az általa használt előfizetéseket, kategóriák szerint csoportosítva akár (pl.: Streaming, Webes szolgáltatások, Játék-beli előfizetések, stb). Ezeket lehetőség szerint valamilyen cloud adatbázisban tárolja (de lehet lokális is), különböző nézetek vannak. Természetesen az asztali klienst több felhasználó is használhatja. A kliens számoljon különböző statisztikákat (havi összköltség, költések megosztása kategóriákra, stb), ezeket szép ábrákon mutassa be.
- **Műsorfigyelő alkalmazás**: Adott domain (pl. színház, mozi, koncert) népszerűbb oldalairól töltse be az alkalmazás az elkövetkezendő eseményeket. Lehessen benne keresni, akár jegyfoglaló oldalra eljutni (ezt böngészőben nyissa meg). Egy egyszerű példa lehet: megadott városok esetén listázza ki, hogy melyik moziban mit játszanak adott napon, erről könnyen értelmezhető listát dobjon ki, esetleg jegyárak feltüntetése mellett. Természetesen szűrhető legyen a lista. A bejelentkezett felhasználó pedig jelölhessen meg eseményeket (tehesse egy adott nevű listára, például "Nyári mozizás 2023"), amiket gombnyomásra exportál pl. Google naptárba, vagy tetszőleges naptári formátumba, ami importálható bárhova. A jelölt elemeket lehessen pdf formátumban is menteni.
- **Színházi "láttam már" alkalmazás**: A felhasználó bejelentkezés után rögzítheti, hogy hol, mikor, milyen darabot látott. Néhány színház esetén (pl.: Szegedi Nemzeti Színház, Szabadtéri, akár eSzínház) oldalak esetén a linket megadva az alkalmazás töltse be a szereposztást, és a darab adatait. Ezekhez készíthessünk különböző statisztikákat, melyik darabot hányféle szereposztásban láttunk, mennyi időt töltöttünk a színházakban egy hónapban/évben, hány rendezőtől láttunk darabot, hány musicalt láttunk, stb. Ezeket menthessük pdf-be. Esetleg a látott darabokat exportálhassuk a Google naptárunkba.

Webes alkalmazás

Követelmények

- Legalább 4 különböző oldaltípus készítése

Hasznos függvénykönyvtárak (csak példának van összeszedve néhány, természetesen bármit lehet használni)

- Flask
- Django
- CherryPy
- Különböző template engine-ek, pl. Jinja

Feladat ötletek

- Minden, ami az asztali alkalmazásnál is jelölve volt.

Chatbot készítése

Követelmények

- Discord/Telegram/Messenger chatbot készítése.
- Legalább 5 különböző interakció**folyam** megvalósítása (tehát nem csak a pontos idő, vagy időjárás lekérése, hanem pl. jegyfoglalás egy nemlétező rendszerbe, a bot kérdezzen vissza, hogy hova kéri a helyet a felhasználó, ha foglalt, ajánljon másikat, stb; személyi asszisztens bot esetén kérje le a Google naptár eseményeit, és szóljon, lehessen új eseményt felvenni, stb.)
- További általános, hasznos interakciók készítése.
- A botot külön be kell mutatni.

Hasznos függvénykönyvtárak (csak példának van összeszedve néhány, természetesen bármit lehet használni)

- https://github.com/topics/discord-chatbot
- discord.py
- python-telegram-bot

Feladat ötletek

- Fentebb említett jegyfoglaló bot (akár ténylegesen foglalhat jegyet egy valós oldalra)
- Személyes asszisztens
- Moderátor bot (reklamáció funkcióval)

Automatizálási feladat

Követelmények

- Valamilyen, egyébként manuálisan elvégezhető feladat elvégzése teljesen (vagy részben)
- Nem csak egy bejelentkezek és lekérem az adatokat jellegű feladatot kell itt elképzelni, hanem komplexebbet.
- Automatizálható tetszőleges hardverigényes feladat is, ezt külön be kell mutatni majd (tehát ha például egy Raspberry Pi + webkamera + mestint kombóval szeretne betörőt detektálni, akkor ha rendelkezésre áll a hardver, akkor ez egy jó feladat lehet), természetesen itt az automatizálás a hardver adatai alapján is történik, de pl. adott eseményeknél küldhet a felhasználónak e-mailt, vagy push üzenetet
- Legalább 4 különböző részfolyamat automatizálása (pl.: bejelentkezés egy weboldalra, szükséges adatok lekérése, adott esetben valamilyen interakció)

Hasznos függvénykönyvtárak (csak példának van összeszedve néhány, természetesen bármit lehet használni)

-

Feladat ötletek

- Betörő észlelése webkamera kép alapján, és levél küldése, lámpa felkapcsolása, stb.
- Részvényekkel kereskedő bot pl.
- Tetszőleges idle webes játékhoz bot készítése

Kutatási feladat

Követelmények

- Valamilyen kutatási feladat megoldása Python nyelv segítségével (pl. adatfeldolgozás/mesterséges intelligencia)
- **Ezt a feladatot egyeztetni kell az oktatókkal!**

Hasznos függvénykönyvtárak (csak példának van összeszedve néhány, természetesen bármit lehet használni)

-

Feladat ötletek

- Képfeldolgozás
- Ajánlórendszer valamilyen problémára
- Priorizáló készítése mesterséges intelligenciával.
    - Pl. Pylint warningok priorizálása a kód/jelentett warning fontossága szerint
- Sentiment elemzés
    - pl. játék chatüzenetekből toxikus játékosok kiszűrése

Játék készítése

Követelmények

- Működő, több szintből álló játék (levelek, nehézség folyamatos növekedése "végtelen" játék esetén).
- Felhasználói interakció.
- Legyen valamilyen globális cél (pl. hercegnő megmentése)
- Ellenfelekkel/különböz puzzle-ökkel
    - Ellenfelek esetén elvárt egy minimális AI (tehát nem csak néz ki a fejéből, hanem próbál harcolni)
    - Puzzle-ök esetén elvárt az egyre növekvő nehézség, a legjobb lenne, ha ez is generált lenne, nem előre beállított.

Hasznos függvénykönyvtárak (csak példának van összeszedve néhány, természetesen bármit lehet használni)

- PyGame
- PyGlet

Feladat ötletek

- 2D platformer
- Chicken Invaders klón
- 2D labirintusos játék generált pályákkal

Pontozás

```
Functionality (8 points)

Core Features (4 points): Does the project meet all the core requirements? Does
it successfully complete the main tasks and functions that it is designed to
perform? This sub-criterion evaluates how well the project fulfills its main
objectives and whether it meets the requirements specified in the project
description.

Error Handling (2 points): Does the project handle errors and exceptions
appropriately? Are error messages clear and helpful? Does it prevent crashes and
unexpected behavior? This sub-criterion evaluates how well the project handles
errors and unexpected events that may occur during its execution.

User Interface (2 points): Is the user interface intuitive and easy to use? Is it
visually appealing and responsive? Does it have good usability features such as
helpful error messages or tooltips? This sub-criterion evaluates how well the
project presents its results to the user and whether it is easy and pleasant to
use.

Object-Oriented Programming (4 points)

Modularity and Organization (2 points): Is the code structured in a modular and
organized manner? Are classes and methods named clearly and appropriately? This
sub-criterion evaluates how well the project is organized and how well it follows
the principles of modular design.

Inheritance and Polymorphism (2 points): Does the project follow OOP principles
such as encapsulation, inheritance, and polymorphism? Are classes and methods
designed in a way that allows for easy extension and modification? This sub-
criterion evaluates how well the project follows the principles of object-
oriented design and whether it allows for easy extension and modification.

Code Quality (4 points)

Readability and Naming (2 points): Is the code well-structured and easy to read?
Are variable and method names descriptive and meaningful? This sub-criterion
evaluates how easy it is to read and understand the code.
```

```
Comments and Documentation (1 point): Is the code properly commented and
documented? Are there clear explanations for the purpose of each function and
class? This sub-criterion evaluates the quality of the comments and documentation
in the code.

Formatting and Styling (1 point): Is the code formatted and styled consistently?
Does it follow PEP8 conventions? This sub-criterion evaluates how well the code
is formatted and styled.

Packaging and Distribution (2 points)

Packaging and Installation (1 point): Is the project packaged in a way that is
easy to install and use by others? Does it have clear installation instructions
and dependencies?

Documentation and Accessibility (1 point): Does the project have a clear and
concise README file? Is the project available on a public repository like GitHub
or PyPI? This sub-criterion evaluates how well the project is packaged and how
accessible it is to others.

Testing (2 points)

Test Coverage (1 point): Does the project have good test coverage? Does it test
all critical parts of the code and handle edge cases properly?

Test Quality (1 point): Are the tests well-structured and easy to read? Do they
cover different use cases and edge cases? This sub-criterion evaluates the
quality of the tests and how well they cover the functionality of the code.

Creativity and Originality (2 points)

Creativity and Originality (2 points): Does the project demonstrate creativity
and originality in its implementation? Does it incorporate additional features or
functionalities beyond the requirements? This sub-criterion evaluates how well
the project goes beyond the basic requirements and demonstrates creativity and
originality in its implementation.
```

Total: 20 points