

Laboratorium 4 – Zmienne środowiskowe, argumenty linii komend, przetwarzanie plików

Cele dydaktyczne

1. Zapoznanie ze zmiennymi środowiskowymi i czytaniem parametrów z linii komend w języku Python.
2. Zapoznanie z uruchomieniem procesów i komunikacji z nimi.
3. Zapoznanie z przetwarzaniem danych w formatach CSV oraz JSON.
4. Zapoznanie z operacjami na systemie plików.

Program można zgłosić jako zrobiony, jeśli spełnione są poniższe warunki:

1. Program jest zgodny z podaną specyfikacją.
2. Został przetestowany.
3. Student go rozumie i potrafi wyjaśnić.

UWAGA:

Wykonywanie zadań przy użyciu czatu GPT będzie traktowane jako praca niesamodzielna i będzie skutkować oceną niedostateczną.

- Zadbaj o to, aby każda funkcja w programie miała tylko jedną odpowiedzialność.
 - Zadbaj o rozdzielenie funkcji przetwarzających dane od funkcji najwyższego poziomu wypisujących tekst na wyjście standardowe.
 - W przypadku, gdy kilka funkcjonalności wymaga skorzystania z samych funkcji, umieść je w osobnym module, który będzie ponownie użyty.
 - Przed potencjalnie źle sformatowanymi danymi zabezpiecz się wykorzystując mechanizm wyjątków.
- **z wykorzystaniem *mechanizmu obsługi wyjątków* zadbaj o poprawność działania funkcji**
 - **do zadań przygotować 3 – 5 testów sprawdzających poprawność działania**
 - **do prowadzącego wysłać pliki z kodem źródłowym oraz rzuty ekranu z przygotowanymi testami**

Wprowadzenie

Zmienne środowiskowe

Istnieją różne sposoby na sterowanie wykonaniem programów komputerowych. Zmienne środowiskowe są zmiennymi, których wartości są ustawiane poza programem, najczęściej przez funkcjonalności wbudowane w system operacyjny albo oprogramowanie zarządzające wykonywaniem usług. Zmienne środowiskowe składają się z par nazwa-wartość. Mogą przechowywać np. konfigurację aplikacji, co jest dobrą praktyką [w tworzeniu aplikacji uruchamianych jako usługi](#).

Innym sposobem sterowania przebiegiem wykonania programu jest wykorzystanie argumentów linii komend. W języku Python dostępne są one na liście [sys.argv](#). Pierwszy argument odpowiada nazwie skryptu, a kolejne reprezentują przekazane parametry. W kolejnych laboratoriach wykorzystane zostaną narzędzia wspierające tworzenie zaawansowanych CLI (ang. command-line interface).

Zadania

Zadanie_1:

Napisz skrypt, który umożliwi uruchomienie go z [dowolną liczbą parametrów linii komend](#):

- 1.1. wyświetla na wyjście standardowe listę wszystkich [zmiennych środowiskowych](#).
- 1.2. Wyświetla na wyjście standardowe podaną listę zmiennych środowiskowych. W takim przypadku, należy przefiltrować zmienne do wyświetlenia na wyjściu standardowym. Warunkiem wyświetlenia zmiennej i jej wartości jest istnienie parametru, którego wartość zawiera się w nazwie zmiennej.
- 1.3. Zmienne powinny być wyświetlone w porządku alfabetycznym.

Punkty: 1

Zadanie_2:

2. Napisz skrypt, który operuje na zmiennej środowiskowej PATH (*zmienna ta wykorzystywana jest w różnych systemach operacyjnych, m.in. Windows, Linux, Mac OS X. Zmienna ta zawiera katalogi, w których znajdują się pliki wykonywalne, które mogą być uruchamiane bez wpisywania pełnej ścieżki do pliku*).

Skrypt powinien umożliwić, z wykorzystaniem samodzielnie ustalonych parametrów linii komend, na realizację poniższych funkcjonalności :

- 2.1. Wypisanie na wyjście standardowe wszystkich katalogów znajdujących się w zmiennej środowiskowej PATH, każdy w osobnej linii.
- 2.2. Wypisanie na wyjście standardowe każdego katalogu znajdującego się w zmiennej środowiskowej PATH wraz z listą wszystkich plików wykonywalnych [znajdujących się w tym katalogu](#).

Punkty: 1

Zadanie_3:

Napisz własną, uproszczoną wersję uniksowego programu `tail`, który będzie wypisywał na wyjście standardowe ostatnie linie zadanego pliku lub danych przekazanych mu na wejście standardowe. Program powinien:

- 3.1. móc być wywołany z argumentem `--lines=n`, gdzie `n` jest liczbą naturalną określającą liczbę linii do wypisania.
- 3.2. móc być wywołany z argumentem `--bytes=n`, gdzie `n` jest liczbą naturalną określającą liczbę bajtów do wypisania.
 - w przypadku wywołania programu bez tych parametrów, program powinien wypisać 10 ostatnich linii.
 - w przypadku, gdy plik ma mniej linii lub bajtów, należy wypisać całą zawartość pliku.
- 3.3. móc być wywołany:
 - przekazując mu danych na wejście standardowe, np.

```
cat plik.txt | python tail.py (w S0 Linux)
```

```
type plik.txt | python tail.py (w S0 Windows)
```

- z argumentem określającym ścieżkę pliku, który ma być wypisany np.

```
python tail.py plik.txt
```

- w przypadku wywołania łączącego te dwa sposoby, np.

```
cat plik.py | python tail.py plik.txt (w S0 Linux)
```

```
type plik.py | python tail.py plik.txt (w S0 Windows)
```

program powinien zignorować dane z wejścia standardowego i wyświetlić dane z pliku.

Punkty: 1

- 3.4. **Wersja rozszerzona (dodatkowo 1 punkt.):** program może dodatkowo przyjąć parametr `--follow`, którego dodanie sprawia, że po wypisaniu zawartości pliku nie kończy działania, lecz czeka na dodanie wierszy do pliku przez inne procesy, a następnie je wyświetla.

Zadanie_4:

3. Napisz program w ulubionym języku programowania (dowolnym, np. C, C++, Rust, Go, Java, Python, PHP, ...), który:
 - 3.1. czyta z wejścia standardowego ścieżkę do pliku tekstowego
 - 3.2. analizuje plik tekstowy pod kątem statystycznym, a następnie oblicza następujące informacje:
 - ścieżka do pliku,
 - całkowita liczba znaków,
 - całkowita liczba słów,
 - liczba wierszy,
 - znak występujący najczęściej,
 - słowo występujące najczęściej.
 - 3.3. wynik obliczeń wypisywany jest na wyjście standardowe powinien w formacie `*.tsv`

3.4. Następnie, napisz skrypt w języku Python, który:

- przyjmuje jako argument linii komend ścieżkę do katalogu w systemie plików,
- z wykorzystaniem modułu [subprocess](#) uruchamia napisany powyżej program do obliczeń, przesyłając na wejście standardowe ścieżki do kolejnych plików,
- przetwarza dane wyjściowe kolejnych wywołań programu, zapisując wynik jako listę słowników,
- wypisuje na wyjście standardowe w dowolnym formacie:
 - liczbę przeczytanych plików;
 - sumaryczną liczbę znaków;
 - sumaryczną liczbę słów;
 - sumaryczną liczbę wierszy;
 - znak występujący najczęściej;
 - słowo występujące najczęściej.

Punkty:3

Zadanie_4:

4. Z wykorzystaniem programów i poleceń wybranego systemu operacyjnego (np. zip, tar, mv, cp, itd.) oraz modułu [subprocess](#) skonstruuuj przedstawione poniżej skrypty. Zadbaj o to, by wspólne funkcjonalności dla obu skryptów zostały wydobyte do niezależnego modułu.

4.1. skrypt **backup.py** do tworzenia kopii zapasowych, który:

- przyjmuje jako argument linii komend ścieżkę do katalogu w systemie plików,
- tworzy archiwum ***.zip** zawierające wszystkie pliki z folderu przekazanego jako argument.
- niech nazwa pliku ma postać `{timestamp}-{dirname}.{ext}`, gdzie:
 - **timestamp** - znacznik czasu zawierający kolejno rok, miesiąc, dzień, godzinę, minutę, sekundę utworzenia pliku,
 - **dirname** oznacza nazwę katalogu,
 - **ext** oznacza rozszerzenie pliku z archiwum.
- przenosi plik do katalogu o nazwie **.backups** w folderze użytkownika. Jeśli ten folder nie istnieje, program go tworzy.
- program pozwala na modyfikację lokalizacji katalogu **.backups** poprzez zmienną środowiskową **BACKUPS_DIR**. Przykładowo, wywołanie programu

```
BACKUPS_DIR=/home/user/alt_backups python backup.py data
```

spowoduje wykonanie kopii zapasowej plików z katalogu `data` oraz zapisanie jej w katalogu `/home/user/alt_backups`.

- dodatkowo, w katalogu **BACKUPS_DIR** skrypt powinien utworzyć plik zawierający historię wykonanych kopii zapasowych.

4.1.1.1. Plik powinien mieć format ***.csv**

4.1.1.2. W pliku powinien zostać zapisany rekord w wybranym formacie zawierający:

- datę wykonania kopii zapasowej,
- pełną lokalizację skopiowanego katalogu
- nazwę pliku z kopią zapasową.

4.1.1.3. W przypadku, gdy plik już istnieje, należy zmodyfikować plik, dopisując rekord do pliku.

4.2. skrypt `restore.py` do przywrócenia katalogu z kopii zapasowej;

- skrypt powinien przyjmować jako argument linii komend ścieżkę do katalogu w systemie plików, **Uwaga:** jeżeli ścieżka nie zostanie jawnie podana, należy przyjąć za ścieżkę [obecny katalog roboczy](#).
- program pozwala na modyfikację lokalizacji katalogu `.backups` poprzez zmienną środowiskową `BACKUPS_DIR`.
- po uruchomieniu, skrypt powinien przeanalizować plik z historią wykonanych kopii zapasowych i wyświetlić użytkownikowi ich ponumerowaną listę w kolejności od najmłodszego do najstarszego.
- użytkownik powinien wskazać wybraną kopię zapasową do przywrócenia, podając jej numer na wejście standardowe.
- po wybraniu kopii, skrypt powinien usunąć zawartość katalogu, a następnie rozpakować archiwum zawierające wybraną kopię zapasową do podanej ścieżki.

Punkty: 4