Kurs Pythona v0.1

Lista zadaniowa IV

Wojciech Adamiec

20 listopada 2022

Spis treści

1 Palindromy

Palindromem nazwiemy wyrażenie brzmiące tak samo czytane od lewej do prawej oraz od prawej do lewej. Przykładowymi palindromami są słowa:

```
kajak
sedes
radar
```

Ale również zdania:

```
Jem sód od ósmej
Mamuta tu mam
Może jutro ta dama da tortu jeżom
```

Napisz procedurę `is_palindrome(text)`, która dla podanego na wejściu tekstu zwróci `True` jeśli tekst jest palindromem oraz `False`, gdy tekst nie jest palindromem.

Uwaga! Zwróć uwagę, że przy sprawdzaniu czy tekst jest palindromem ignorujemy całkowicie spacje oraz nie zwracamy uwagi na wielkość liter.

Wskazówka: Zapoznaj się z działaniem funkcji `split`, `join` oraz `lower`.

2 Podział

Bardzo użyteczną metodą dla napisów w pythonie jest metoda `split`, która dzieli napis ze względu na blok białych znaków (o ile nie podamy żadnego opcjonalnego argumentu tej metodzie). Przykładowe wywołanie metody `split`:

```
"  Hobbit   ma   kota ".split() -> ["Hobbit", "ma", "kota"]
```

Napisz procedurę `my_split(text)`, która będzie działała analogicznie (z dokładnością do nieco innego interfejsu). Twoja procedura powinna przyjąć napis oraz zwrócić listę słów tak jak robi to metoda `split`.

Przykładowe wywołanie procedury `my_split`:

```
my_split("  Hobbit   ma   kota ") -> ["Hobbit", "ma", "kota"]
```

W implementacji procedury `my_split` **nie wolno** Ci użyć metody `split`. Dla uproszczenia jako blok białych znaków będziemy traktować tylko blok spacji.

3 MinMax

Napisz procedurę `min_max(data)`, która przyjmuje listę liczb `data` oraz zwróca parę liczb `minimum, maximum` (które będą odpowiednio najmniejszą liczbą z listy oraz największą liczbą z listy). Przykładowe wywołania procedury `min_max`:

```
min_max([2, 1, 3, 7]) -> (1, 7)
min_max([-5, 3, 3, 7, 12, -2, 0, 12, 1]) -> (-5, 12)
```

W implementacji procedury `min_max(data)` **nie wolno** Ci użyć procedur wbudowanych `min` oraz `max`.

Jeśli masz ochotę możesz postarać się zoptymalizować swoją procedurę, tak aby nie wykonywała więcej porównań niż jest to absolutnie konieczne. Najbardziej optymalny algorytm powinien wykonywać co najwyżej $\lceil \frac{3}{2}n - 2 \rceil$ porównań (gdzie `n = len(data)`).

4 Granica

Przeanalizuj i przetestuj dostarczoną przez prowadzącego procedurę `info` służącą do drukowania na terminalu (w bardziej czytelnej formie niż `print`) zawartości macierzy dwuwymiarowej rozmiaru $a \times b$.

Następnie napisz procedurę `border_map(a, b)`, która zwraca macierz (listę list) o wymiarach $a \times b$ (gdzie a jest współrzędną poziomą, b pionową) postaci:

```
XXXXXXXXXXXXXXX
X.............X
X.............X
X.............X
X.............X
X.............X
X.............X
XXXXXXXXXXXXXXX
```

Powyższy obrazek powstał poprzez wyrażenie `info(border_map(15, 8))`.

Oczywiście sama procedura `border_map` zwraca jedynie macierz i niczego sama nie drukuje. Przykładowe wywołanie procedury `border_map(4, 3)` powinno zwrócić macierz:

```
[['X', 'X', 'X', 'X'], ['X', '.', '.', 'X'], ['X', 'X', 'X', 'X']]
```

Można ją oczywiście zwizualizować za pomocą wyrażenia `info(border_map(4, 3))`:

```
XXXX
X..X
XXXX
```

Elementami macierzy powinny być napisy `"X"` oraz `"."`

5 Rysowanko

W repozytorium znajdują się pliki z danymi do obrazków postaci:

```
232,217,198   232,217,198   233,218,199   234,219,200   ...
232,217,198   232,217,198   233,218,199   233,218,199   ...
232,217,198   232,217,198   233,218,199   233,218,199   ...
232,217,198   233,218,199   233,218,199   234,219,200   ...
...           ...           ...           ...           ...
```

Dane należy czytać następująco: W j-tym wierszu, w i-tej kolumnie znajduje się trójka liczb opisująca wartości kanałów RGB (z przedziału $0 - 255$) piksela o współrzędnych (i, j).

W pliku do zadania znajduje się procedura `get_image_data_from_file(file)`, która z danego na wejściu pliku `file` wczytuje dane dotyczące pikseli, a następnie zwraca macierz (listę list), której elementami są trójki liczb (r, g, b) z wartościami odpowiadającymi liczbom z wczytanego pliku.

Spróbuj przeanalizować dostarczony kod (nie przejmuj się, jeśli nie będziesz go dokładnie rozumieć), a następnie dokładnie przeanalizuj co zwraca wywołanie dostarczonej procedury dla dowolnego z dostępnych plików (np. `get_image_data_from_file("image_data_1.txt")`).

Będziemy chcieli na podstawie danych z pliku narysować obrazek za pomocą modułu `turtle`. Każdy piksel z danych będziemy wizualizować jako kwadrat niewielkiego rozmiaru.

Sugerując się komentarzami z pliku do zadania narysuj za pomocą kwadratów o odpowiednim kolorze i pozycji obrazek bazując na wczytanych z pliku danych.

Operowanie bezpośrednio na pikselach (pikselach z żółwia, a nie pikselach wczytanych z pliku, które będziemy traktować jak małe kwadraty), tak jak robiliśmy to do tej pory, jest bardzo niewygodne. Dlatego chcemy wprowadzić swój własny, nowy układ współrzędnych - nazywany czasem siatką (ang. *grid*). Zanim przystąpisz do implementowania procedury `square(x, y, colour)` (która przyjmuje współrzędne `x`, `y` będące wyrażone już w naszym układzie współrzędnych) musisz uzupełnić implementację procedury `to_pixels(x, y)`, która dla podanych współrzędnych w naszym układzie zwróci parę współrzędnych w układzie pikseli.

Uwaga! Program nie będzie działał bez wywołania `colormode(255)`, które zmienia format wartości kanałów RGB z $0 - 1$ na $0 - 255$.

Wskazówka: Do debugowania warto używać animacji żółwia z wysoką prędkością. Animacja jest jednak zdecydowanie za wolna, aby narysować cały obrazek w sensownym czasie. Kiedy upewnisz się, że wszystko powinno działać - użyj wyrażeń `tracer(0, 1)` oraz `update()`.