Efektywne programowanie w języku Python, 2019/2020
Projekt A16, Pokaz slajdów – Dominika Kapanowska

Głównym zadaniem mojego programu jest stworzenie pokazu obrazków z danego katalogu. Biorę pod uwagę pliki z rozszerzeniem *.jpg* i *.png*.
Wykorzystuję do tego moduł *tkinter* i bibliotekę graficzną *PIL*.
Obrazki są wyświetlane w kolejności alfabetycznej, w jakiej występują w katalogu. Istnieje możliwość zdefiniowania własnej kolejności, poprzez umieszczenie w danym katalogu pliku o nazwie *order.txt*, zawierającego w kolejnych liniach nazwy wszystkich obrazków w pożądanej kolejności. Jeśli plik z kolejnością jest niepoprawny, zostaje zignorowany, a kolejność jest alfabetyczna.
Istnieje również możliwość dodania opisu / komentarza do obrazka.
W katalogu projektu znajdują się obrazki *start.jpg* i *end.jpg*, są one dodawane niezależnie, jest to slajd tytułowy i kończący.

1. Uruchamianie programu

Główny program znajduje się w pliku *project.py*, można go uruchomić na dwa sposoby:

a) podając ścieżkę do katalogu w argumentach programu, a następnie:

```
run_project(curr_dir())
```

Funkcja `curr_dir()` sprawdza, czy został podany argument programu, a następnie, czy taki katalog istnieje. Jeśli nie, to zwraca pusty stringa, jeśli tak, to zwraca tę ścieżkę.

b) podając jako argument bezpośrednio ścieżkę do katalogu (string):
`run_project(sciezka_do_katalogu)`

Sprawdzeniem, czy `sciezka_do_katalogu` jest poprawna, zajmuje się funkcja `run_project`, którą omówię później. Funkcja `run_project` zwraca string, więc zaleca się uruchomić program poleceniem `print(run_project)`.

2. Działanie funkcji pomocniczych

a) `init_pictures(curr_dir, order_needed)`

`curr_dir` – ścieżka do katalogu, z którego robię pokaz slajdów
`order_needed` – wartość logiczna, czy był plik *order.txt*

Najpierw tworzę listę plików o rozszerzeniu *.jpg* i *.png w* curr_dir (kolejność alfabetyczna).
Jeśli nie ma zdefiniowanej kolejności, zwracam tę listę, w przeciwnym razie czytam plik *order.txt* liniami, i tworzę listę tych linii. Jeśli ma ona te same elementy, co ta pierwsza, to oznacza, że plik *order.txt* jest poprawny, zwracam ją. Jeśli elementy są różne, to ignoruję kolejność i zwracam pierwszą listę.

b) `mult(tuple_arg, m)`

Mnożę każdy element krotki przez stałą m, konwertuję do liczby całkowitej i zwracam to w postaci listy. Wykorzystane później w funkcji `wanted_size`.

c) `get_name(path)`

Wyciągam ze ścieżki nazwę samego pliku / katalogu.

d) `wanted_size(tuple_arg)`

Argumentem jest rozmiar obrazka, przeskalowuję go, żeby ładnie wyglądał w moim okienku, zwracam nowe wymiary jako listę. Nie powiększam obrazków, żeby nie tracić na jakości. Mogę tylko zmniejszać.

e) `reformat_comment(comment)`
Funkcja ta przyjmuje jako argument komentarz do obrazka (string). Okienko w pokazie slajdów jest dość wąskie, więc w tym miejscu komentarz jest dzielony na linie długości równej maksymalnie 14 znaków.

f) `load_comments(dir_from)`
Funkcja ta przeszukuje katalog podany jej jako argument, czy jest tam plik *comments.txt*, liczy również, ile jest w nim obrazków, któę będą użyte do slideshow. Gdy nie ma pliku *comments.txt*, funkcja zwraca listę w pustymi stringami w ilości o 2 większej, niż liczba obrazków (będzie jeszcze sjajd tytułowy i kończący, któe nie będą skomentowane). Jeśli zaś plik się istnieje, to funkcja czyta go liniami, i każdą linie dodaje do rezultatu końcowego, który jest listą tych komentarzy. Linia, czyli komentarz do obrazka, najpierw zostaje jeszcze sformtowana funkcją `reformat_comment`, aby miała odpowiednią szerokość.
Wymagania co do pliku *comments.txt*: plik musi zawierać tyle linii, ile jest obrazków przewidzianych do pokazu slajdów, linie mogą być puste, komentarze muszą odpowiadać kolejności obrazków w pokazie slajdów.

3. Głowna klasa *App*

Klasa ta dziedziczy po klasie Tk z modułu tkinter. Rozmiar okna i obrazków w nim jest dostosowany do ekranu. Jako kolor domyślny tła jest kolor biały.
Jako argument podajemy listę ścieżek obrazków(`image_files`), którą utworzyliśmy wcześniej.
Okienko składa się z dwóch części:
- panelu z przyciskami, polem tekstowym i miejscem na komentarz
- panelu z miejscem na wyświetlany obrazek
`self.id` - indeks wyświetlanego właśnie obrazka.
Dla każdego uruchomienia tworzę plik z historią pokazu, omówię to później.

a) `browse_files(self.image_files)`

`image_files` to lista ścieżek do obrazków przekazana jako parametr konstruktora, ręcznie dodaję do niej obrazek startowy na początek i kończący na koniec.

Uzupełniam listę `self.pictures`, tworząc obrazki za pomocą biblioteki *PIL*, przeskalowuję wcześniej wymiary.
Pokaz slajdów rozpoczynam od wyświetlenia obrazka startowego, a potem już używam przycisków.

b) przyciski

Każdy przycisk ma swoją pozycję, wymiary, tekst i przypisaną komendę.

PREV – zmniejsza o 1 `self.id` i wyświetla obrazek. Jeśli nie ma już wcześniejszych, wyświetla messagebox z informacją.
NEXT – zwiększa o 1 `self.id` i wyświetla obrazek. Jeśli nie ma już następnych obrazków, wyświetla messagebox z informacją i zamyka okienko z pokazem.
FIRST – ustawia `self.id` na 1 i wyświetla pierwszy (po startowym) obrazek.
LAST – ustawia `self.id` na `len(self.pictures) - 2` i wyświetla ostatni (przed końcowym) obrazek.
QUIT – natychmiast kończy pokaz slajdów poprzez zamknięcie okienka.
AUTO – powoduje, że obrazki przełączają się same z opóźnieniem 1 sekundy.
DFLT – ustawia domyślną kolorystykę okienka.
SET – współpracuje z `self.text_box` – omówione niżej.

c) `self.text_box`

W moim okienku znajduję się pole tekstowe. Można wpisać w nie nazwę lub kod koloru, w jakim chce się mieć okienko. Poprawność koloru sprawdzam za pomocą funkcji `is_color_like` z biblioteki `matplotlib`. Żeby to zrobić, używam przycisku SET. Jeśli kolor nie jest poprawny, wyświetlam messagebox z informacją, jeśli jest poprawny, ustawiam go jako motyw okienka. Pole tekstowe jest czyszczone automatycznie.

d) `self.comment_label`
Jest to obiekt klasy `tk.Label`, w którym wyświetlam komentarz do aktualnie pokazywanego obrazka. Komentarze są przechowywane w liście `self.comments`, któa łądowana jest na początku w kontruktorze za pomocą funkcji `load_comments`.

e) `self.picture_display`

Miejsce na wyświetlany w danej chwili obrazek.

4. `run_project(curr_dir)`

Główna funkcja do uruchomienia pokazu.
Najpierw następuje sprawdzenie, czy ścieżka do katalogu nie jest pusta lub czy on istnieje. Jeśli te warunki nie są spełnione, kończymy program i zwracamy string z informacją

```
"Wrong name of a directory! / There is nothing to show!"
```

To zachodzi również wtedy, gdy w danym katalogu nie ma żadnych obrazków do pokazania.

Następnie sprawdzamy, czy w danym katalogu jest wspomniany wcześniej plik *order.txt*, jeśli tak ustawiamy zmienną `order_needed` na `True`, inaczej na `False`. Zmienną tą oraz ścieżkę do katalogu przekazujemy funkcji `init_pictures`.
Jeśli były jakieś obrazki, to tworzymy obiekt klasy `App`, następnie tworzymy plik z historią.
Nazwy tych plików maja postać:

```
'history' + str(int(time.time())) + '.txt'
```

dzięki czemu łatwo można je odróżniać, plik pokazu odpalonego na końcu będzie się znajdował na dole w katalogu `History`.
Każde przełączenie obrazku na inny powoduje zapisanie nazwy tego obrazka do pliku z historią. Pomijamy obrazek startowy i końcowy.
Po zamknięciu pokazu zwracam stringz informacją:

```
"Project run successfully, check the " + history_name + " file."
```

gdzie `history_name` to nazwa pliku z historią dla tego pokazu slajdów.

5. Testowanie

Mój projekt zawiera unittesty w pliku *tests.py*. Do testowania służy klasa `Test` dziedzicząca po klasie `TestCase` z modułu `unittest`. Testowanie odbywa się następująco:

a) testowanie działania funkcji pomocniczych (mnożenia krotki, wyciągania nazwy katalogu / pliku ze stringa, tworzenie listy ścieżek w zależności od kolejności lub bez)
b) sprawdzenie, czy dla katalogu bez obrazków, o złej ścieżce, lub gdy nie ma nazwy katalogu, program zwraca string `"Wrong name…"`
c) sprawdzenie historii pokazu, gdy nie ma w katalogu pliku z kolejnością
d) sprawdzenie historii pokazu, gdy w katalogu jest plik z kolejnością
e) sprawdzenie historii pokazu, gdy w katalogu jest plik z kolejnością, ale jest on nieprawidłowy
f) sprawdzenie, czy funkcja `is_color_like` działa poprawnie
g) sprawdzenie, czy funkcja `reformat_comment` działa poprawnie

W celu przejścia testów należy, gdy pojawią się okienka, przejść pokazy slajdów (klikając przycisk NEXT lub AUTO). Wtedy linijka w programie głównym (*project.py*) nie powinno znajdować się żadnych wywołań funkcji (na czas testowania należy je wykomentować).