Projektrapport - ASCII Art studio - Uppgift 1

Programmet

Programmet låter dig ladda en bild och rendera den till ascii-art. Programmet har inget eget UI som man kan agera med utan körs endast via din terminal. ASCII Art studio startas genom att köra pyton-filen DA2005-Projekt-Ascii.py i din terminal.

Moduler, klasser och bibliotek

DA2005_Projekt_Ascii.py - Här importeras två huvudsakliga bibliotek och en egen testklass.

Image - Image från PIL (Python Imaging Library) används för all bildbehandling i programmet. För att installera PIL/Pillow kör du kommandot `pip install Pillow` i terminalen, där Pillow är den moderna versionen av PIL som aktivt underhålls. Detta bibliotek är inte en del av Pythons standardbibliotek, så det måste installeras separat.

unittest - Standardbibliotek. Pythons inbyggda testramverk. Detta används för att köra enhetstester automatiskt när programmet startar. Unittest är en del av Pythons standardbibliotek.

TestAsciiArtProject - Egen testklass från filen Test_Project.py

Test_Project.py - Här importeras 6 av pythons standardbibliotek, Image (näms även ovan) och huvudprojektet DA2005_Projekt_Ascii som testerna ska göras på.

unittest - Standardbibliotek för enhetstester.

os - Standardbibliotek som ger tillgång till operativsystemsfunktioner, särskilt för filhantering som `os.path.exists()` och `os.unlink()` som används för att kontrollera om temporära testfiler finns och för att radera dem efter att testerna är klara.

tempfile - Standardbibliotek som används för att skapa temporära filer och mappar som automatiskt rensas bort efter användning. "tempfile.NamedTemporaryFile()" används för att skapa testbilder som kan användas i tester.

io - Detta bibliotek tillhandahåller verktyg för att arbeta med input/output-strömmar, inklusive "io.StringIO()" som skapar en minnesbuffert som fungerar som en fil. Används för att fånga utskrifter från funktioner för att testa att rätt felmeddelanden skrivs ut.

sys - Systembibliotek som ger tillgång till systemspecifika parametrar och funktioner, bla. “sys.stdout” som är standardutgången. Används för att omdirigera utskrifter till din StringIO-buffert så att du kan testa vad som skrivs ut.

PIL.Image - Används för att skapa testbilder till testerna

unittest.mock.patch - Avancerat testverktyg som temporärt ersätter funktioner eller objekt med "mock"-objekt under testkörning. Används för att simulera fel och kontrollera att funktioner hanterar undantag korrekt.

DA2005_Projekt_Ascii - Egen huvudmodul som innehåller alla funktioner för ASCII-art-projektet som testas.

Struktur

Huvudprogrammet - ***DA2005_Projekt_Ascii.py***

- Interaktivt konsolprogram med menysystem
- Funktionsbaserad struktur
- Kärnfunktioner:
    - "load_image()" - Laddar bilder med hjälp av PIL
    - "render_ascii_image()" - Konverterar bild till ASCII-art
    - "build_ascii_art()" - Bygger ASCII-strängen från pixeldata
    - "info()" - Visar bildinformation
    - "prompt_for_image()" - Användarinteraktion för bildladdning
    - "main()" - Huvudloop med 5-alternativ meny

Testfil - **Test_Project.py**

- Testklass, "TestAsciiArtProject(unittest.TestCase)"
- Testinfrastruktur:** Temporära testbilder, stdout-mocking
- Testmetoder:
    - Bildladdning (framgång/fel)
    - ASCII-rendering (olika scenarier)
    - Felhantering och exceptions
    - Edge cases och input validation

Reflektioner

Load hantering

Jag resonerade lite fram och tillbaka kring hur jag på bästa sätt skulle hantera loadfunktionaliteten. Till en början så hade jag ingen load funktion då det jag bara hade 2 rader:

img = Image.open(filename)

img.load()

Efter att jag la till felhanteringen så bröt jag ut ovan 2 rader tillsammans med felhanteringen till funktionen load_image() som vid success returnerar ett image objekt och som vid exception returnerar None.

I funktionen main() så skapade jag en while loop som skulle loopa tills användaren lyckats mata in ett korrekt filnamn som korrekt laddas.

Denna while loop bröt jag också ut till funktionen promt_for_image() då jag tyckte att det blev en cleanare kod i main funktionen,

Ascii art hantering

Jag valde först att ha en funktion för detta men valde sedan att dela upp den i 2 funktioner, en funktion render_ascii_image() som hanterar konvertering av bilden till gråskala, korrigera bildens mått och hämta en lista med gråskalan för varje pixel i bilden. Funktionen kallar sedan på build_ascii_Art() med dessa värden som returnerar Ascii art strängen.

Felhantering

Det var en utmaning att avväga felhanteringen om när man ska använda try/exception, raise och IF-satser och samtidigt inte överdriva med implementeringen av detta.

Ex: Jag har en funktion render_ascii_image() som tar fram variabler som skickas vidare som argument för funktionen build_ascii_art(). I detta fall så har jag använt if satser för att säkerställa att build_ascii_art() inte anropas med felaktiga argument.

Build ascii art har därmed endast ett try/exception block som förväntas fånga oväntade fel.

I funktionen load_image() som är direkt beroende av användar input så vill jag att användaren ska få ett mer informativt meddelande om vad som har gått fel så att användaren kan förändra sin input.

Testning

Testningen är det enda som är utvecklat helt obejktorienterat. Denna bit blev snabb rörig då jag skapade större delen av koden via Github CO-pilot. Jag lyckades efter flera vändor fram och tillbaka ta bort och lägga till vad jag tycker är relevanta tester. Jag antar att best practice bör vara att hellre testa för mkt än för lite och testar därmed ganska mycket i min mening. Jag testar exempelvis alla särskilda exeptions som jag fångar upp i load_image() funktionen och alla IF-satser jag har i render_ascii_image() funktionen.