Project: File Organizer

Path: start2impact University Master Data Science Course 2 Python e Numpy

Author: Giacomo Abramo

Programmare non significa conoscere tutte le funzioni e le librerie di un linguaggio: ogni giorno ti troverai a che fare con costrutti nuovi. Prima di pensare a come integrarli nel tuo codice

- studia la documentazione (puoi farlo offline da dentro Jupyter): quali sono gli argomenti di una funzione? Cosa restituisce? Quali sono le funzioni e gli strumenti di una libreria?
- cerca esempi di utilizzo sul Web (StackOverflow è il primo dei tuoi migliori amici)
- apri Jupyter e fai degli esperimenti: prova la nuova libreria/funzione nel caso più semplice, isolato dal contesto della tua applicazione.

Il problema che devi affrontare è difficile? Impara a smontarlo in sottoproblemi. Per esempio: prima di pensare a un ciclo prova a implementare una singola iterazione in una cella del tuo notebook. In questo progetto ti chiederò di spostare dei file in una sottocartella a seconda della loro estensione. Come si crea una sottocartella? Come si sposta un singolo file? Come si estrae l'estensione dal nome di un file? Una volta che hai affrontato questi task secondari, pensa a come integrarli nella tua applicazione.

Jupyter è il secondo dei tuoi migliori amici. Non sai quanto vale una variabile? Vuoi testare solo alcune righe di codice o una singola operazione? Apri una cella del tuo notebook e isola il (sotto)problema che vuoi risolvere. Le shortcut di Jupyter velocizzano molto(!) la scrittura del tuo codice. Perdi dieci minuti per imparare quelle fondamentali, e familiarizzaci per qualche giorno. Vedrai che ne vale la pena.

Ricordati che ogni operazione (il controllo di una condizione, l'apertura di un file, l'aggiornamento di una struttura dati) ha un costo. Una volta che hai preparato i pezzi del tuo codice, pensa a come organizzarli nell'ottica di minimizzare il suo costo totale. Quando scrivi un ciclo, per esempio, pensa a quali operazioni conviene

eseguire fuori dal ciclo (prima o dopo) e quali invece devono essere necessariamente ripetute ad ogni iterazione.

Commenta sempre il codice che scrivi, a beneficio degli altri e di te stesso dopo una settimana, quando non ti ricorderai niente di quello che hai scritto e perché.

Non ci capisci più niente? Prenditi una pausa (idealmente, riposa gli occhi un minuto ogni venti minuti e stacca completamente un quarto d'ora ogni due ore). Fai altro e dimenticati del codice. Prima di ricominciare, riavvia il notebook.

Adesso entriamo nel vivo del progetto.
Le **specifiche** sono la parte principale di qualsiasi progetto. Assicurati di aver letto bene cosa ti chiedo di fare. Per esempio:
- quanti file devi consegnare e in che formato?
- come li devi chiamare?
- cosa devono contenere le righe del file di recap degli Step 1 e 2?
- quante cifre decimali hanno i valori della tabella riassuntiva dello Step 3?
- …

Cerca sempre di rispettare le specifiche assegnate, evitando di complicati la vita se non esplicitamente richiesto. Questo è un consiglio Zen.

Step 1

Il **path** di un file è il suo percorso sul *file system*. Un path *assoluto* è il percorso dalla cartella *root*, cioè la cartella "più in alto" della gerarchia. Su Windows, per esempio, i path assoluti partono da *C*. Un path *relativo*, invece, si riferisce alla cartella corrente. In generale, **cerca di evitare i path assoluti**. Visto che il percorso di un file sarà diverso da utente a utente, se qualcun altro vuole eseguire il tuo codice dovrà cambiarli (tutti) a mano. In questo progetto puoi supporre che il notebook degli Step 1 e 3 sia allo stesso livello (cioè nella stessa cartella) della cartella *files/* che contiene i file da spostare. Potrai quindi fare riferimento a questa in modo relativo. Un'alternativa, se ti torna meglio, è spostarti fisicamente nella cartella files/; puoi farlo attraverso la funzione *os.chdir*.

Le specifiche chiedono di **aggiornare il file di recap**. Sembra una richiesta semplice, ma molti studenti si perdono qualche pezzo. Prima di buttarti sul codice, prova a chiederti:

- com'è fatto il file di recap?

- quali operazioni devono essere eseguite se il file non esiste?
- e se invece esiste già?

Per poter lavorare tutti i file dentro la cartella files/, dovrai scrivere un *ciclo*. Ricordati che tutte le operazioni nel corpo del ciclo verranno eseguite ad ogni iterazione del ciclo stesso. Chiediti quindi **quali, tra le logiche che dovrai implementare** (gestione e scrittura del file di recap, creazione delle sottocartelle, stampa delle informazioni dei file, spostamento dei file...), **meritino di essere inserite dentro al ciclo** (e quindi essere eseguite per ognuno dei file), e quali invece è sufficiente eseguire una sola volta, prima o dopo il ciclo.

Step 2

Come si legge nelle specifiche, **dovrai scrivere il codice di questo Step in un file .py, e dotarlo di una *CLI***, cioè una *interfaccia a linea di comando*.

Attenzione alle differenze rispetto allo Step 1. Qui **l'obiettivo è spostare un singolo file**, il cui nome è inserito dall'utente a linea di comando. Che succede se il file passato dall'utente non esiste? È sempre necessario un ciclo su tutti i file dentro files/?

Quando sviluppo un piccolo script Python in formato .py, testo i singoli pezzi con Jupyter (che è comodissimo per prototipare) assemblandoli via via in un'unica cella, prima di spostarli in un file separato. Questo è il mio personale modo di lavorare, e non è detto che sia comodo per tutti. Un'alternativa è usare un editor pensato per lavorare direttamente con il formato .py: è un buon momento per dare un'occhiata a PyCharm.

La **libreria *argparse*** (anche se esistono altri strumenti, come *input* e *sys.argv*) è lo standard per creare script con una CLI. Ti chiedo di provare a usarla, è una libreria molto utile e flessibile. Consigli:

- quando usi una libreria nuova, provala prima nel caso più semplice che ti viene in mente, isolato dal contesto della tua applicazione; integrala nel codice dell'applicazione, insieme alle altre logiche, solo quando la padroneggi
- leggi attentamente la *documentazione* delle funzioni che usi: che argomenti ha, per esempio, la funzione *add_argument* di *ArgumentParser*?

Step 3

Puoi supporre che lo Step 1 sia già stato lanciato, e che quindi **esista già una cartella *images* con dentro qualche immagine**. Niente vieta di testare il codice di questo Step con ulteriori immagini.

Attenzione a una cosa. Come da specifiche, **gli Step 1 e 3 fanno cose diverse in momenti diversi**: non vuoi sapere cosa c'è dentro la cartella images ogni volta che sposti un file. Non c'è quindi motivo di accorparli. In generale, se una funzionalità non è richiesta dalle specifiche, è inutile complicarsi la vita per implementarla.

La libreria *NumPy* è fondamentale in questo percorso. Anche se sei interessato solo alla programmazione Python, NumPy è comunque uno strumento molto prezioso. In breve: in questo progetto non puoi farne a meno. **Per ogni immagine, ti chiedo di usare NumPy per:**

- **capire le dimensioni** (altezza, larghezza)
- **capire il tipo** (grayscale, RGB, RGBA)
- **calcolare la media dei valori dei pixel.**

Prima di pensare al ciclo che dovrai scrivere, inizia lavorando un'immagine alla volta. Attraverso la funzione *np.array*, puoi trasformare l'immagine in un array NumPy. Un array NumPy ha tre proprietà fondamentali: *ndim*, *shape* e *size*.

Prova quindi a chiederti:

- quante dimensioni (proprietà *ndim*) ha un'immagine in scala di grigio?
- e una RGB? Quale sarà la sua *shape*?
- se voglio aggiungere anche la trasparenza (il canale A), di quante nuove dimensioni ho bisogno? Come cambia *ndim*? Come cambia la *shape*?

Puoi usare queste domande come traccia per

- organizzare il controllo di flusso (*if-elif-else*)
- implementare il codice di questo Step usando (solo) NumPy.

NumPy ha una funzione *mean* molto efficiente. Inutile (re)implementare con Python il calcolo della media. Piuttosto, chiediti come utilizzare in modo furbo l'argomento *axis* della funzione mean.

Per la libreria *tabulate*, con cui dovrai costruire la tabella riassuntiva, valgono consigli simili a quelli che ti ho dato per argparse: prima di integrarla nel codice dello Step, provala da sola. In questo caso Jupyter ti sarà molto utile.