Pattern Recognition
and Applications Lab

La Programmazione ad Oggetti in Python

Docente: Ambra Demontis

Anno Accademico: 2024 - 2025

Corso di Laurea in Ingegneria Elettronica, Informatica e delle Telecomunicazioni

University of Cagliari, Italy

Department of Electrical and Electronic Engineering

La Programmazione ad Oggetti in Python

In queste slide vedremo oggetti e funzioni per leggere e scrivere file in Python.

Leggere e Scrivere File

Quando si implementa un programma è spesso necessario **salvare dei dati in file** o **leggere** da essi dei **dati**.

In Python esistono oggetti e funzioni built-in ad-hoc per questo scopo.

La Funzione Open

Per poter leggere o scrivere file, dobbiamo prima di tutto **creare un oggetto di tipo file**.

Per fare questo, si utilizza la funzione built-in *open.*

A questa funzione devono essere passati due argomenti:

- Il **percorso** del file che vogliamo leggere/scrivere
- La **modalità di apertura**

La sintassi è

<nome_variabile> = open(“percorso_file”, “modalità_di_apertura”)

Il Percorso

Il percorso può essere:

- **Assoluto** cioè che contiene tutta la sequenza dei nomi delle directory a partire dalla directory radice del file system.
- **Relativo** cioè che fa riferimento alla posizione nel file system dello script che state eseguendo.

Il Percorso Assoluto

La **sintassi con la quale vanno scritti** dipende dal sistema operativo.

Ad esempio, se il nome del file che vogliamo creare è dati.txt.

Su Windows, se volessimo crearlo nella directory C:\Users\Anna\, il percorso assoluto sarebbe "C:/Users/Anna/dati.txt"
(notate che è uguale ma con / al posto di \).

Mentre su Linux o Mac OS, se volessimo crearlo nella home dell'utente Anna il percorso assoluto sarebbe: "/home/Anna/dati.txt"

Il Percorso Relativo

Il percorso relativo fa riferimento alla posizione dello script che state eseguendo.

Se vogliamo riferirci ad un file nella stessa directory dello script possiamo usare la sintassi: “<nome_file>”

Se invece per riferirci alla **directory padre** di quella in cui si trova lo script possiamo usare: “**../**<nome_file>”

I “../” possono anche essere concatenati per risalire nella gerarchia delle cartelle. Es: “../../<nome_file>”

La Modalità di Apertura

La modalità di apertura può essere:

- “r” Se vogliamo **leggere** un file esistente
- “w” Se vogliamo **scrivere** un nuovo file.

  Se esiste già un file con quel nome verrà sovrascritto da un file vuoto.
- “a” Se vogliamo **appendere** (aggiungere) dei dati in un file esistente.

  Se il fine non esiste ancora viene creato.

La Funzione Open

Supponiamo di voler scrivere il file chiamato “dati.txt” nella stessa directory nella quale si trova il nostro script.
Supponiamo di voler aprire l’oggetto di tipo file in modalità di scrittura.

Il codice che dovremo scrivere sarà:

```
nuovo_file = open("dati.txt", "w")
```

Questa funzione creerà un oggetto di tipo file e lo memorizzerà nella variabile chiamata nuovo_file.

Il Metodo Write

A questo punto possiamo **scrivere** sul file utilizzando il metodo dell'oggetto file chiamato ***write***.

Ad esempio, il codice:

```
nuovo_file = open("dati_utente_idx_3.txt", "w")
nuovo_file.write("Anna")
```

Prima creerà un file chiamato dati_utente_idx_3.txt nella stessa cartella nella quale si trova lo script che contiene questo codice.
Poi, andrà a scriverci la stringa "Anna".

Il Metodo Write

E’ possibile chiamare più volte il metodo write in modo da aggiungere dei caratteri nel file

Ad esempio, se nel codice precedente aggiungessimo:

```
nuovo_file = open("dati_utente_idx_3.txt", "w")
nuovo_file.write("Anna")
nuovo_file.write(" ")
nuovo_file.write("Bianchi")
```

Nel file verrà scritto:
Anna Bianchi

A Capo

Se vogliamo che il testo venga scritto a capo, possiamo utilizzare "\n"

Esempio:

```
nuovo_file = open("dati_utente_idx_3.txt", "w")
nuovo_file.write("Anna\nBianchi")
```

Nel file verrà scritto:

Anna

Bianchi

Il Metodo Close

Appena abbiamo terminato di scrivere dati nel file dobbiamo **chiudere il file** richiamando il metodo ***close*** dell'oggetto file.

```
nuovo_file = open("dati_utente_idx_3.txt", "w")

nuovo_file.write("Anna")
nuovo_file.write(" ")
nuovo_file.write("Bianchi")

nuovo_file.close()
```

Il Metodo Close

Supponete di avere un programma lungo e di dover scrivere su un file sia all'inizio che alla fine del programma.

Dobbiamo aprire il file a inizio programma e chiuderlo quando abbiamo finito di scrivere tutto, quindi quando il programma termina? No!

Dobbiamo tenere aperto il programma solo quando è necessario per compiere delle operazioni di lettura scrittura.

Nel caso di esempio caso dovremmo aprirlo ad inizio programma, scrivere, chiuderlo e ripetere la stessa sequenza di operazioni a fine programma.

Il Metodo Close

Perchè bisogna evitare di tenere i file aperti?
Le ragioni sono molteplici.

Alcune di queste sono:

1) Alcuni sistemi operativi hanno un “numero massimo” di file che possiamo tenere aperti contemporaneamente
2) Se scriviamo su un file i dati potrebbero non venire effettivamente scritti fino a quando non chiudiamo il file, che è ovviamente un problema se degli altri programmi hanno bisogno dei dati che stiamo scrivendo.

Il Metodo Close

Provate a lanciare da Pycharm il seguente script:

```
import time

nuovo_file = open("dati_utente_idx_3.txt", "w")
nuovo_file.write("Anna")
nuovo_file.write(" ")
nuovo_file.write("Bianchi")
print("Scritto")
time.sleep(10000)
print("Finito")
```

Questo script scrive su file Anna Bianchi e poi rimane in esecuzione “dormiente” per simulare un programma “lungo”.

(Quando il programma termina in Python tutti i file da esso aperti vengono automaticamente chiusi).

Il Metodo Close

Quando vedete a schermo la stampa “scritto” ma prima di vedere quella “finito”, potreste non vedere il file dati_utente_idx_3.txt o vederlo vuoto.

Nonostante la funzione write sia già stata invocata.

Scrivere Dati su un File

Per evitare di dimenticare di richiamare il metodo close, generalmente si usa una sintassi differente da quella mostrata nelle slide precedenti.

```
with open(<percorso_file>, <modalità_apertura>) as <variabile>:
    <variabile>.write(<stringa>)
```

Con questa sintassi viene creato un oggetto di tipo file con la funzione open e viene assegnato alla variabile <variabile>.

Poi vengono eseguite tutte le operazioni rientrate rispetto alla prima riga.

Infine, il file viene automaticamente chiuso.

Scrivere Dati su un File

Esempio:

```
with open("dati_utente_idx_3.txt", "w") as nuovo_file:
  nuovo_file.write("Anna")
  nuovo_file.write(" ")
  nuovo_file.write("Bianchi")
```

Context Manager

Come mai con questa sintassi il file viene chiuso automaticamente?

La classe che definisce gli oggetti di tipo file (quelli che creiamo con la funzione open e che utilizziamo per compiere operazioni sui file) ha due metodi speciali:
__enter__
__exit__

Gli oggetti appartenenti a classi che implementano questi metodi vengono detti **context manager** e **con questi oggetti possiamo utilizzare l'istruzione with**.

Context Manager

```
with open(<percorso_file>, <modalità_apertura>) as <variabile>:
    <variabile>.write(<stringa>)
```

Prima di effettuare le istruzioni indentate rispetto alla prima riga Python richiama il metodo __enter__, definito dalla classe file.

Poi esegue tutte le istruzioni indentate.

Quando non ci sono più istruzioni indentate da eseguire viene richiamato il metodo __exit__ definito dalla classe file, che si occupa di chiudere il file.

Il Metodo Readlines

Per **leggere** dati da un file si può utilizzare il metodo dei file chiamato ***readlines***.

Questo metodo restituisce un oggetto iterabile.
L'oggetto iteratore di questo oggetto iterabile, ad ogni chiamata della funzione next restituisce una riga del file.

Il Metodo Readlines

Se nel file dati_utente_idx_3.txt ci fosse scritto:
Anna
Bianchi

```
with open("dati_utente_idx_3.txt", "r") as nuovo_file:
  for l in nuovo_file.readlines():
     print("riga:", l)
```

Stamperebbe:
riga: Anna

riga: Bianchi

Il Metodo Readlines

NB: La riga Anna contiene il carattere di “a capo”.
Se volessimo eliminarlo potremmo farlo utilizzando il metodo split delle stringhe.

Specificare il Percorso in Modo Opportuno

Negli esempi di lettura/scrittura da file abbiamo sempre letto/scritto file nella stessa cartella nel quale si trova lo script che stiamo eseguendo.

Nei progetti reali questo spesso non accade.

Spesso si utilizzano infatti cartelle apposite per salvare i dati in modo che siano separati dal codice.

Specificare il Percorso in Modo Opportuno

Supponete di avere la seguente struttura di package e moduli:

progetto_ecommerce
- main.py
- ecommerce
  - __init__.py
  - database.py
  - prodotti.py
  - pagamenti
    - __init__.py
    - pagamenti.py
- dati
  - fatture

Supponete di avere una cartella “dati” dove volete salvare i vostri file e che nello script pagamenti, vogliate scrivere dei file nella cartella “fatture”.

Come scrivete il percorso per riferirvi alla cartella “fatture” nel vostro script?

Specificare il Percorso in Modo Opportuno

Con ciò che abbiamo visto le scelte possibili sono:

1. Scrivere come stringa il percorso assoluto, es:
   "/home/Anna/progetto_ecommerce/dati/fatture"

1. Scrivere il percorso relativo:
   "../../../"

Tuttavia queste scelte causano alcuni problemi...

Specificare il Percorso in Modo Opportuno

Scrivere come stringa il percorso assoluto, es:
"/home/anna/progetto_ecommerce/dati/fatture"

Supponete di spostare il progetto dentro un'altra cartella… dovreste cambiare tutti i percorsi.

Inoltre il codice **non potrà essere utilizzato da terzi** (a meno che non abbiano una struttura delle directory, a partire dalla radice del file system fino alla directory contenente il progetto, identica alla vostra).

Specificare il Percorso in Modo Opportuno

Scrivere il percorso relativo:

```
"../../../"
```

Secondo la struttura dei package, il percorso potrebbe risultare molto lungo e quindi poco leggibile.

Specificare il Percorso in Modo Opportuno

Per risolvere questi problemi in modo semplice si può utilizzare un percorso assoluto ma, invece che ricavarlo manualmente e scriverlo come stringa nel codice, **si può fare in modo che venga calcolato**.

In questo modo se cambia il percorso del progetto non è necessario aggiornare percorsi che puntano alla directory “data”.

La classe Path

Per far questo possiamo sfruttare la classe ***Path***, definita dalla libreria ***pathlib***.

Questa classe definisce dei metodi utili per ricavare i path.

La classe Path

Sfruttando questo metodo, possiamo ad esempio inserire nella directory “dati” uno script (e.g., chiamato “path.py”).

In questo script:

1) Recuperiamo il percorso assoluto della cartella padre dello script path.py (la cartella “dati”) utilizzando la funzione dirname della libreria path.
2) Creiamo un oggetto di tipo path che contenga quel percorso.

```
import os
from pathlib import Path
PATH_DATA_DIR = Path( os.path.dirname(__file__)  )
```

La classe Path

Dallo script pagamenti.py potremo importare questa costante e utilizzare il metodo ***joinpath*** della classe Path per creare il percorso del file che vogliamo leggere o scrivere.

Il metodo joinpath aggiunge al path tutte le stringhe che gli vengono passate come argomento.

La classe Path

```
from copy import deepcopy
from dati.path import PATH_DATA_DIR
file_path = PATH_DATA_DIR.joinpath("fatture", "fattura_idx_3.txt")
print(file_path)
```

Stamperebbe:
/home/anna/progetto_ecommers/dati/fatture/fattura_idx_3.txt

La classe Path

```
from copy import deepcopy
from dati.path import PATH_DATA_DIR

file_path_idx3 = deepcopy(PATH_DATA_DIR)
file_path_idx3.joinpath("fatture", "fattura_idx_3.txt")
print(file_path)
```

Se vogliamo evitare di modificare l'istanza che contiene il path della directory "data"

Stamperebbe:

/home/anna/progetto_ecommers/dati/fatture/fattura_idx_3.txt

Esercizio sulla Lettura di Dati da un File

Considerare il progetto GestioneNegozio del quale avete ricevuto il link tramite il google group.

Creare nel package “gestione_clienti” tutte le classi necessarie per leggere da un file, salvando temporaneamente nel programma, al suo avvio, i dati dei clienti che sono stati salvati su un file.

Ricordatevi di applicare il paradigma della programmazione orientata agli oggetti. Quali classi vi servono?

Il file nel quale si trovano è il file chiamato “dati_clienti.txt”, nella cartella dati.
La cartella “dati” supponiamo sia fatta per contenere tutti i dati del progetto memorizzati su file.

Esercizio sulla Lettura di Dati da un File

I dati del cliente supponiamo abbiano il formato:
nome;cognome;CF;n_tel

Dove CF è il codice fiscale e n_tel è il numero di telefono

Esercizio sulla Lettura di Dati da un File

Nel package “dati” inseriremo un modulo “path” che conterrà:

```
import os
from pathlib import Path
PATH_DATA_DIR = Path( os.path.dirname(__file__)  )
```

Esercizio sulla Lettura di Dati da un File

Nel package “gestione_clienti” creeremo un modulo “c_cliente”

```
class CCliente():

 def __init__(self, nome, cognome, CF, telefono):
   self.nome = nome
   self.cognome = cognome
   self.CF = CF
   self.telefono = telefono

def stampa_dati_cliente(self):
 print("nome: {}, cognome:{}, CF:{}, numero_telefono:{}".format(
   self.nome,self.cognome,self.CF,self.telefono))
```

Esercizio sulla Lettura di Dati da un File

Nel package “gestione_clienti” creeremo un modulo “c_clienti”

```
from .c_cliente import CCliente
From copy import deepcopy

class CClienti():

 def __init__(self, path_dati):
   self._clienti = []
   self._path_file_clienti = deepcopy(path_dati)
   self._path_file_clienti = self._path_file_clienti.joinpath("dati_clienti.txt")
   self._carica_clienti_da_file()

… continua
```

Esercizio sulla Lettura di Dati da un File

```
def _carica_clienti_da_file(self):

 with open(self._path_file_clienti, "r") as f_clienti:
   for riga in f_clienti.readlines():
     #rimuoviamo il carattere a capo
     riga_divisa = riga.split("\n")[0]

     #dividiamo in base al carattere separatore
     riga_divisa = riga_divisa.split(";")
     nome = riga_divisa[0]
     cognome = riga_divisa[1]
     CF = riga_divisa[2]
     n_telefono = riga_divisa[3]

     cliente = CCliente(nome, cognome, CF, n_telefono)
     self._clienti.append(cliente)
```

Esercizio sulla Lettura di Dati da un File

```
def stampa_dati_clienti(self):
 for cliente in self._clienti:
   cliente.stampa_dati_cliente()
```

Esercizio sulla Lettura di Dati da un File

Per provare a vedere se effettivamente i dati vengono caricati possiamo creare un modulo “main” dentro la cartella principale del progetto.

```
from gestione_clienti.c_clienti  import CClienti
from dati.path import PATH_DATA_DIR
from copy import deepcopy

clienti = CClienti( PATH_DATA_DIR)

clienti.stampa_dati_clienti()
```

Esercizio sulla Scrittura di Dati su File

Modificate le classi precedentemente create mettendo a disposizione degli utilizzatori delle stesse un metodo “aggiungi_cliente” che permetta ad un utente di aggiungere da tastiera un nuovo cliente.

Quando il nuovo cliente verrà aggiunto i suoi dati dovranno essere anche scritti sul file in modo che possano essere caricati da esso al prossimo avvio del programma.

Esercizio sulla Scrittura di Dati su File

```
from .c_cliente import CCliente

from dati.path import PATH_DATA_DIR

class CClienti():

    …

    def _memorizza_cliente(self, ogg_cliente):
     with open(self._path_file_clienti, "a") as f_clienti:
       f_clienti.write(ogg_cliente.nome + ";"+ ogg_cliente.cognome +
                       ";"+ ogg_cliente.CF + ";"+ ogg_cliente.telefono+"\n")

    def aggiungi_cliente(self):
     cliente = CCliente.crea_cliente()
     self._clienti.append(cliente)
     self._memorizza_cliente(cliente)
```

Esercizio sulla Scrittura di Dati su File

```
class CCliente():

    …

    @classmethod
    def crea_cliente(cls):
     nome = input("inserisci il nome del cliente ")
     cognome = input("inserisci il cognome del cliente ")
     CF = input("inserisci il codice fiscale del cliente ")
     telefono = input("inserisci il numero di telefono del cliente ")
     return cls(nome, cognome, CF, telefono)
```

Esercizio sulla Scrittura di Dati su File

```
from gestione_clienti.c_clienti  import CClienti
from dati.path import PATH_DATA_DIR

clienti = CClienti( PATH_DATA_DIR)

clienti.stampa_dati_clienti()

clienti.aggiungi_cliente()
clienti.stampa_dati_clienti()
```