Laboratorio d'Informatica

Corso di Laurea Magistrale in Ingegneria per l'Ambiente e il Territorio

A.A. 2021/2022

Docente: Lorenzo Putzu

Lezione 06

Il linguaggio Python

Sequenze nidificate

- Dato che gli elementi di una lista possono essere valori di tipi qualsiasi, possono essere quindi anche dei dati strutturati, come stringhe o altre liste.
- L'operatore di indicizzazione consente di accedere anche agli elementi di strutture nidificate.
- Se s è una variabile a cui è stata assegnata una lista, e l'elemento di indice *i* è a sua volta una sequenza (lista o stringa), sarà possibile accedere all'elemento di indice *j* di quest'ultima con la seguente **sintassi**:

```
s[i][j]
```

- Una delle possibili applicazioni delle liste è la rappresentazione di vettori e analogamente, le liste nidificate possono essere convenientemente usate per rappresentare **matrici**, cioè entità bidimensionali.

Liste nidificate

- È possibile rappresentare una matrice con una lista nidificata, in modo da ottenere una corrispondenza diretta e intuitiva tra gli indici di riga e colonna di un elemento della matrice e quelli dello stesso elemento della lista.
- Questo risultato si ottiene considerando una matrice come una **sequenza ordinata** di *m* righe, ciascuna delle quali è a sua volta una **sequenza ordinata** di *n* valori semplici (numeri reali). Ogni riga può quindi essere rappresentata con una lista di *n* numeri, mentre l'intera matrice sarà rappresentata da una lista di *m* liste (righe della matrice).
- In questo modo l'elemento della matrice nella *i*-esima riga e nella *j*-esima colonna corrisponderà all'elemento della lista avente indici *i* - 1 e *j* - 1.

$$\begin{pmatrix} -3 & 1 & 4 \\ 2 & 5 & -1 \end{pmatrix} = \quad [[-3, 1, 4], [2, 5, -1]]$$

Esercizi

1. Scrivere una fuzione che data una matrice memorizzata in una lista nidificata, stampi il contenuto della matrice
2. Stampare la tavola pitagorica dei numeri da 1 a 10 sotto forma di una matrice:

```
1   2     3     ...   9     10
2   4     6     ...   18    20
...             ...         ...
10  20    30    ...   90    100
```

3. Definire una funzione che acquisisca come argomento una lista L e restituisca una nuova lista contenente tutti gli elementi che compaiono in L **una sola volta**. Esempio: [3, 1, 5, 1, 6, 2, 9, 6] → [3, 5, 2, 9]

Dizionari

- Le liste consentono di rappresentare dati strutturati i cui valori siano composti da una **sequenza ordinata** di valori più semplici.
- Un altro caso comune nella pratica è quello in cui i valori dei dati da elaborare siano rappresentabili come **collezioni** (insiemi) di valori più semplici e **non ordinati**, ciascuno dei quali abbia un significato che possa essere descritto con un **nome simbolico**.
- Un esempio sono le informazioni anagrafiche su una persona come nome, cognome, data e luogo di nascita, codice fiscale, ecc.
- I dati aventi tali caratteristiche possono essere rappresentati in Python per mezzo del tipo strutturato dizionario.

Dizionari

- Il tipo di dato *dizionario* è costituito da collezioni non ordinate di un numero qualsiasi di elementi, ciascuno dei quali è composto da un valore di un tipo **qualsiasi**, e da una chiave che lo identifica univocamente

- In altre parole, un dizionario è un **insieme** di **coppie** chiave–valore non ordinate, alle quali si può accedere solo attraverso la chiave corrispondente.

- Le chiavi devono essere scelte dal programmatore, e possono essere valori numerici o Booleani, oppure nella maggior parte dei casi si tratta di stringhe. Nella scelta delle chiavi è buona norma usare simboli mnemonici.

Dizionari

- Le sequenze di coppie chiave–valore possono essere definite in maniera schematica racchiudendole tra parentesi **graffe** e separate tra loro dalla virgola ‘**,**’ mentre in ogni coppia, la chiave e il valore sono separati dal carattere ‘:’
- Sintassi

```
{chiave1:valore1, chiave2:valore2, . . . }
```

- Ad esempio un dizionario che contiene informazioni anagrafiche su una persona: nome, cognome ed età può essere definito come

```
{"nome":"Maria", "cognome":"Bianchi", "eta":28}
```

- Un dizionario che contiene le coordinate di un punto nel piano cartesiano

```
{"x":-1.2, "y":3.4}
```

- Un dizionario che contiene una data (giorno, mese e anno)

```
{"giorno":1, "mese":6, "anno": 2017}
```

- infine un dizionario vuoto:

```
{}
```

Dizionari

- Anche i valori di tipo *dizionario* costituiscono **espressioni** Python. È quindi possibile:
  - scrivere un dizionario nella *shell*: dopo la pressione del tasto INVIO, il dizionario verrà mostrato nella stessa *shell*
  - stampare un dizionario con l'istruzione print, per esempio:

    ```
    print({"x":-1.2, "y": 3.4})
    ```
  - assegnare un dizionario a una variabile, per esempio:

    ```
    punto = {"x":-1.2, "y": 3.4}
    ```
  - acquisire un dizionario attraverso la tastiera, per esempio:

    ```
    d = eval(input("Inserire un dizionario: "))
    ```
- **Nota**: quando un dizionario viene mostrato nella *shell* le coppie chiave/valore compaiono in un ordine che non può essere controllato dal programmatore.

Dizionari

- Così come per le liste, anche i valori degli elementi di un dizionario possono essere indicati attraverso espressioni. Per esempio, dopo l'esecuzione della seguente sequenza di istruzioni:

```
n = "Maria"
c1 = "Ros"
c2 = "si"
a = 5
persona = "nome":n, "cognome":c1 + c2, "età":a**2
```

- la variabile persona sarà associata al seguente dizionario: "nome":"Maria", "cognome":"Rossi", "età":25

Dizionari

- Inoltre, anche i valori degli elementi di un dizionario possono appartenere a un tipo qualsiasi, e quindi possono essere valori strutturati come stringhe, liste e dizionari (nidificati).
- Per esempio, il dizionario seguente contiene il nome, il cognome e la data di nascita di una persona (giorno, mese e anno):

```
persona = {"nome":"Maria", "cognome":"Rossi", "g":1, "m":1, "a":1995}
```

- Le stesse informazioni possono essere memorizzate in una forma più strutturata con due dizionari nidificati:

```
data = {"g":1, "m":1, "a":1995}
persona = {"nome":"Maria", "cognome":"Rossi", "data_nascita":data}
```

Dizionari: indicizzazione

- Anche per i dizionari esistono diversi operatori e funzioni predefinite.
- L'operatore principale è quello che consente l'accesso ai **valori** dei singoli elementi. Il nome (*indicizzazione*) e la sintassi sono identici a quelli dell'analogo operatore per le liste, con la differenza che gli elementi di una lista sono **ordinati** e quindi identificati univocamente dalla loro posizione (indice), mente gli elementi di un dizionario non hanno un ordine predefinito ma sono identificati dalla loro chiave.
- L'operatore di indicizzazione consente di accedere al **valore** di ogni elemento di un dizionario, per mezzo della **chiave** corrispondente.
- **Sintassi**:

```
dizionario[chiave]
```

- dizionario deve essere un'espressione che produca un dizionario (di norma, il nome di una variabile)
- chiave deve essere un'espressione il cui valore corrisponda a una delle chiavi del dizionario

Dizionari: indicizzazione

- Il risultato dell'operatore di indicizzazione è il valore dell'elemento di dizionario associato a chiave. Se il valore di chiave non corrisponde a una delle chiavi del dizionario si otterrà un messaggio d'errore.
- L'operatore di indicizzazione consente anche di:
  - **modificare** il valore associato a una chiave **esistente**
  - aggiungere una **nuova** coppia chiave–valore, la cui chiave **non** sia presente nel dizionario
- **Sintassi**:

```
dizionario[chiave] = valore
```

- Se chiave fa parte di dizionario, il valore corrispondente viene sostituito da valore; altrimenti viene aggiunto al dizionario l'elemento chiave:valore

Dizionari: costruzione

- Questo vuol dire che possiamo costruire un dizionario attraverso l'operatore di indicizzazione, a partire da un dizionario vuoto.
- In molti programmi è necessario costruire un dizionario avente un insieme di chiavi **predefinito** dal programmatore, alle quali dovranno essere associati valori acquisiti durante l'**esecuzione** del programma.
- Ad esempio quando è necessario memorizzare i dati di un nuovo utente (che non conosce il linguaggio Python) possiamo demandare all'utente stesso di inserire i propri dati e memorizzarli un dizionario opportuno. Ad esempio con la seguente sequenza di istruzioni

```
persona = {}
print("Inserire i seguenti dati anagrafici:")
persona["nome"] = input("Nome: ")
persona["cognome"] = input("Cognome: ")
persona["età"] = eval(input("Età: "))
print("I dati inseriti sono:", persona)
```

Dizionari: operatori di confronto

- Gli operatori == e != consentono di scrivere espressioni **condizionali** (il cui valore sarà True o False) consistenti nel confronto tra due dizionari.
- **Sintassi**:

```
dizionario1 == dizionario2
dizionario1 != dizionario2
```

- dove dizionario1 e dizionario2 indicano **espressioni** che abbiano come valore un dizionario.
- Il risultato della prima espressione condizionale è vero (quindi i dizionari sono considerati identici) se contengono le stesse **coppie** chiave/valore, **indipendentemente** dal loro ordine.
  - {"x":-1.2, "y":3.4} == {"y":3.4, "x":-1.2} → True

Dizionari: in e not in

- Questi operatori consentono di scrivere espressioni **condizionali** che hanno lo scopo di verificare se una certa chiave sia presente o meno all'interno di un dizionario.
- **Sintassi**:
  - espressione in dizionario
  - espressione not in dizionario
- dove espressione indica una qualsiasi espressione Python e se il **valore** di espressione è presente tra le **chiavi** del dizionario l'operatore `in` produce il valore True; in caso contrario produce False. Il comportamento dell'operatore `not in` è quello opposto.

Dizionari nidificati

- Si è già detto che anche i valori associati alle chiavi di un dizionario possono appartenere a qualsiasi tipo di dato, e possono quindi essere a loro volta dati strutturati, cioè stringhe, liste e dizionari.
- Gli elementi di dati strutturati contenuti in un dizionario sono a loro volta accessibili attraverso l'operatore di indicizzazione, con la stessa sintassi già descritta per il caso di strutture nidificate all'interno di una lista.
- In particolare, se d è una variabile a cui è stato assegnato un dizionario, e l'elemento d[chiave] è un valore strutturato (una sequenza o un altro dizionario), sarà possibile accedere all'elemento di tale valore avente indice (se si tratta di una sequenza) o chiave(se si tratta di un dizionario) k con la seguente **sintassi**:

```
d[chiave][k]
```

Dizionari: funzioni predefinite

- Alcune metodi e funzioni predefinite di utilità generale sono le seguenti:
  - `len(dizionario)`
    restituisce la lunghezza di un dizionario
  - `dizionario.keys()`
    restituisce le chiavi di un dizionario
  - `dizionario.values()`
    restituisce i valori di un dizionario
  - `dizionario.items()`
    restituisce gli elementi di un dizionario come un insieme di coppie chiave/ valore
  - `dizionario.pop(key)`
    se esiste elimina e restituisce il valore dell'elemento con chiave key, altrimentirestituisce un errore

Dizionari: funzioni iterative

- Come per le liste anche per i dizionari è possibile usare l'istruzione `for` accedendo iterativamente a **tutte** le chiavi del dizionario
- Sintassi

```
for k in d :
    sequenza di istruzioni
```

- k deve essere il nome di una variabile che non sia già usata per memorizzare dati all'interno dello stesso programma
- d deve essere un'espressione avente come valore un dizionario
- sequenza di istruzioni è una sequenza di una o più istruzioni qualsiasi che di norma eseguono un'operazione sulla variabile k

Dizionari: funzioni iterative

- Tuttavia, a differenza delle liste k contiene la chiave del dizionario perciò questo significa che sarà possibile modificare i suoi elementi assegnando nuovi valori a d inserendo nella sequenza di istruzioni un assegnamento del tipo

  d[k]= espressione

- dove espressione è una qualsiasi espressione del linguaggio python
- E' comunque possibile eseguire delle iterazioni sui dizionari richiamando contemporaneamente la chiave e il valore, sfruttando il metodo `items`

  for k, v in d.items() :

    sequenza di istruzioni

- k come prima contiene le chiavi del dizionario d
- v contiene una **copia** del valore corrispondente in d

Strutture dati

- La scrittura di un programma si compone di due fasi fondamentali:
  - la formulazione di un algoritmo
  - la scelta delle strutture dati per rappresentare i dati da elaborare

- La scelta delle strutture dati più opportune è un passo fondamentale in quanto influenza la complessità del programma che si scriverà o ne pregiudica la codifica.

- Volendo memorizzare un singolo utente (come visto prima) la scelta di un dizionario potrebbe essere sufficiente, ma dovendo memorizzare *n* utenti tale scelta diventerebbe troppo dispersiva, dato che sarebbero necessarie *n* variabili di tipo dizionario
  - *inoltre n* potrebbe non essere noto a priori
  - l'uso di istruzioni iterative non sarebbe possibile

Strutture dati

- Per memorizzare i dati di un utente avrei potuto utilizzare anche una lista, dato che può contenere anche dati eterogenei

- Un dizionario però è preferibile in questo caso in quanto ad ogni valore corrisponde un'etichetta chiara che consente infatti **al programmatore** di ricordare facilmente il significato di ciascun elemento attraverso la chiave corrispondente, purché si usino chiavi mnemoniche

- L'uso di una lista invece richiederebbe una costante attenzione per ricordare (o consultare la porzione di codice in cui è definita la lista) a quale componente equivale un determinato indice della lista
- persona_a = {"nome":"Ada", "cognome":"Neri", "età":25}
- persona_b = ["Ada", "Neri", 25]

Esercizi

4. Definire una funzione che riceva come argomenti tre numeri rappresentanti una data (giorno, mese e anno) e restituisca un dizionario contenente i tre valori associati alle chiavi `"giorno"`, `"mese"` e `"anno"`.
   Esempio: `{"giorno":8, "mese":10, "anno":2019}`

5. Definire una funzione che, dato un numero di persone, acquisisca nome, cognome ed età di ciascuna di esse e memorizzi tali dati in una lista di dizionari aventi chiavi `"nome"`,`"cognome"`,`"eta'"`.
   Esempio:
   `{"nome":"Ugo", "cognome":"Gui", "eta'":25}`