Corso ITS: *ARTIFICIAL INTELLIGENCE SPECIALIST*

Modulo: **Programmazione Procedurale in Python**

Docente: *Andrea Ribuoli*

Giovedì **30 Gennaio** 2025

09:00 - 14:00

---

verifica esercizio assegnato

- il **contenitore** di tipo *lista* è **solo uno** dei tipi di contenitore
- Python offre anche **insiemi** e **dizionari**
- potremo **combinare** più contenitori per ottenere strutture più complesse

insiemi

- un **insieme**(*set*) memorizza valori **univoci**
- ossia non possono esistere elementi duplicati
- contrariamente alle *liste*, non esiste un ordine degli elementi
- quindi, non vi si può accedere con un indice
- le **operazioni** sugli insiemi sono quella della matematica
- *maggiore velocità* rispetto alle liste (non occorre ordinamento)

In [30]:
```
PICCHE = chr(9824)
FIORI  = chr(9827)
CUORI  = chr(9829)
QUADRI = chr(9830)
semi = {  PICCHE, FIORI, CUORI, QUADRI }
print(semi)
```

```
{'♣', '♥', '♦', '♠'}
```

- l'**ordine** di visita dipende dalla modalità di memorizzazione (dettagli tecnici)

In [31]:
```
for seme in sorted(semi) :
```

```
    print(seme)
```

♠
♣
♥
♦

- non si può inizializzare un **insieme vuoto** in modo simile alle liste
- è necessario scrivere: `insieme_vuoto = set()`

In [32]:
```
print(len(semi))
```

4

- operatore di appartenenza è **in**

In [33]:
```
if PICCHE in semi :
    print("Ci sono anche le PICCHE")
```

```
Ci sono anche le PICCHE
```

- si possono *aggiungere* (`add`) e *rimuovere* (`discard` .vs. `remove`) elementi
- **discard**, se non trova l'elemento, lascia l'insieme invariato ma non segnala errore
- **remove**, se non trova l'elemento, solleva una *eccezione*
- il metodo **clear** svuota l'insieme di tutti gli elementi presenti

- concetto di **sotto-insieme**
- metodo **issubset**: *True* o *False*
- insiemi uguali? `==`
- insiemi diversi? `!=`
- concetto di **unione**
- metodo **union** (l'operazione elimina eventuali duplicati)
- concetto di **intersezione**
- metodo **intersection**
- concetto di **differenza**
- metodo **difference**

dizionari

- un **dizionario**(*dictionary*) è un contenitore che memorizza associazioni **chiave-valore** (*key-value*)
- ogni chiave è associata ad un valore (univoche)
- un singolo valore può però essere associato a più chiavi

In [34]:
```
famiglia = { "Andrea": 62, "Laura": 59, "Roberto": 34, "Giovanni": 29, "Fran
```

- coppie chiave-valore separate dai **due punti** (`:`)
- elenco raccolto tra **parentesi graffe** (`{}`)
- coppie separate da **virgole** (`,`)
- come con **list()** per le liste con **dict()** si copia un dizionario
- l'operatore di indicizzazione (`[key]`) si usa per accedere al valore associato

In [35]:
```
print("Giovanni ha già compiuto", famiglia["Giovanni"], "anni")
```

```
Giovanni ha già compiuto 29 anni
```

- dizionario **vuoto**: `nipoti = {}`
- modifica valore con `famiglia["Francesca"] += 1`
- eliminazione coppie con `pop(key)` (eccezione se key assente)
- scansione delle chiavi con `for key in famiglia :` (ordine in base a ottimizzazione)
- alternativamente `for key in sorted(famiglia :` (se chiave alfanumerica, ordine lessicografico)
- scansione dei valori con `for eta in famiglia.values() :`
- in un colpo solo: `for item in contact.items() :` (**items** restituisce sequenza di tuple)