

# 3. Strutture dati in Python

Corso di Python per il Calcolo Scientifico

# Outline

- Pile e code
- List comprehension
- Tuple
- Set
- Dizionari

# Pile e code

- Il linguaggio Python offre metodi su liste per implementare in maniera automatica le pile.
- In particolare, abbiamo il metodo **append()** che ci permette di inserire un nuovo elemento in cima alla lista, mentre il metodo **pop(pos)** estrae l'elemento in posizione **pos**. Di default, **pos** è pari a **len(lista) - 1**.

```
s = [1, 2, 3]
s.append(4)
e = s.pop()
```

# Pile e code

- Python ci permette di implementare in maniera semplificata anche le code.
- Usiamo il metodo `pop(0)` per l'operazione di estrazione, ed il metodo `insert(pos, el)` per inserire l'elemento `el` in posizione `pos`, con `pos` pari a 0.

```
q = [1, 2, 3]
q.insert(0, 4)
e = q.pop(0)
```

# Pile e code

- **Esercizio 1:** proviamo a valutare il tempo necessario alle operazioni di **insert** e **pop** su una coda in Python usando la libreria **time**. Confrontiamo il risultato ottenuto con quello ottenibile implementando una coda come una struttura di tipo **deque** e usando gli opportuni metodi **appendleft** e **popleft**.

# List comprehension

- La **list comprehension** è una delle tecniche più usate per creare delle liste.
- Sostituisce, per questo tipo di operazioni, i cicli.
- La forma classica è la seguente:

```
lista_output = [f(elemento) for elemento in lista_input]
```

- Si può anche specificare una forma estesa con un'istruzione condizionale:

```
lista_output_if = [f(elemento) for \  
    elemento in lista_input if condizione]
```

# List comprehension

- **Esercizio 2:** selezioniamo tutti i nomi che iniziano con la lettera **B** dalla seguente lista. Facciamolo usando un ciclo ed una list comprehension.

```
lista_nomi = [  
    "Jax Teller",  
    "Walter White",  
    "Billy Butcher",  
    "Luke Skywalker",  
    "Bobby Singer",  
    "Johnny Lawrence"]
```

# List comprehension

- **Esercizio 3:** ottenere una lista che abbia al suo interno tutti i quadrati dei numeri che vanno da 1 a 10.
- **Esercizio 4:** ottenere una lista che abbia la stringa **pari** in corrispondenza dei numeri pari, mentre quella **dispari** in corrispondenza dei numeri dispari, per tutti i numeri che vanno da 1 a 10.



# Tuple

- Una **tupla** rappresenta un insieme di valori eterogenei **ripetibili**.
- Sono simili alle liste, ma hanno metodi differenti e, soprattutto, **non sono mutabili**.
- Una tupla può contenere liste, tuple, o anche altre strutture dati.

```
tupla = ('hello', 'world', [1, 2, 3])
```

# Set

- I **set**, o **insiemi**, sono anch'essi molto simili alle liste.
- Tuttavia, hanno una caratteristica importante: **non possono esserci valori ripetuti**.
- Hanno una sintassi di questo tipo:

```
insieme = { 1, "stringa", 2 }
```

- Si possono utilizzare per isolare tutti gli elementi presenti in una lista, evitando le ripetizioni.

# Dizionari

- I **dizionari** sono dati da una serie di **coppie chiave – valore**.
- In ciascuna coppia, vi è una chiave, che funziona come ‘indice’, ed un **valore** accessibile.
- La chiave può essere un intero o una stringa; il valore può essere un valore qualsiasi.
- Le chiavi non possono essere ripetute, visto che servono per l’accesso al singolo elemento.

# Dizionari

- È possibile creare un **dizionario vuoto**:

```
dizionario = {}
```

- ed inserire delle coppie chiave – valore:

```
dizionario['k'] = 'v'
```

```
dizionario[1] = 'n'
```

- Per accedere ad un valore associato ad una determinata chiave:

```
dizionario[1]
```

# Dizionari

- Per recuperare tutte le chiavi:

```
chiavi = dizionario.keys()
```

- Per recuperare tutti i valori:

```
valori = dizionario.values()
```

- Per recuperare tutte le coppie chiave – valore:

```
coppie = dizionario.items()
```

# Dizionari

- Per creare un dizionario non vuoto, possiamo usare il metodo **dict()**:

```
dizionario = dict(k1=1, k2=2)
```

- Possiamo usare il metodo **zip()** per creare un dizionario da due liste:

```
chiavi = ['k1', 'k2']
```

```
valori = [1, 2]
```

```
dizionario = dict(zip(chiavi, valori))
```

- Esiste anche la **dict comprehension**:

```
output = {  
    chiave: valore for \  
    valore in iterabile }
```

# Dizionari

*Esercizio 5: scrivere una dict comprehension che permetta di ottenere il dizionario **vecchio\_o\_giovane** dato il seguente dizionario:*

```
dizionario = {  
    'Jax Teller': 27,  
    'Walter White': 52,  
    'Billy Butcher': 41,  
    'Luke Skywalker': 79,  
    'Bobby Singer': 68,  
    'Johnny Lawrence': 49}
```

*In particolare, il dizionario **vecchio\_o\_giovane** avrà le stesse chiavi del dizionario di partenza, a cui sarà associato il valore **giovane** soltanto se il valore della chiave del dizionario di partenza è inferiore a 65.*

# Domande?

42