Il file `qualificate.txt` contiene le 48 squadre qualificate ai mondiali di calcio del 2026, ordinate in ordine alfabetico e ciascuna associata ad un punteggio. Di seguito si riporta un estratto del file, limitatamente (per brevità) alle prime 3 e alle ultime 3 squadre.

1: File qualificate.txt

```
Algeria,1564.26
Argentina,1874.81
Australia,1580.67
...
USA,1673.13
Uruguay,1673.07
Uzbekistan,1465.34
```

Il punteggio viene usato per stilare un ranking: valori maggiori indicano squadre più forti. Per il sorteggio dei gironi, le squadre devono essere suddivise in fasce sulla base dei seguenti criteri:

- la squadra (o le squadre) degli stati ospitanti sono inserite nella prima fascia;
- tutte le altre squadre sono inserite nelle fasce in base al ranking: le squadre con punteggio più alto completano la fascia 1, le successive vengono inserite nella fascia 2, e così via fino alla fascia 4.

La suddivisione in fasce ottenuta in base al contenuto del file `qualificate.txt` (considerando che gli stati ospitanti sono USA, Mexico e Canada) è la seguente.

Fascia 1	**Fascia 2**	**Fascia 3**	**Fascia 4**
USA	Germany	Australia	Uzbek.
Mexico	Croatia	Algeria	Qatar
Canada	Colombia	Egypt	Iraq
France	Senegal	Norway	S.Africa
Spain	Uruguay	Panama	SaudiArabia
Argentina	Japan	CIV	Jordan
England	Swiss.	Sweden	Bos.Herz.
Portugal	Iran	Paraguay	CaboVerde
Brazil	Turkiye	Czechia	Ghana
Neth.	Ecuador	Scotland	Curacao
Morocco	Austria	Tunisia	Haiti
Belgium	KoreaRep.	DRCongo	NewZealand

Table 1: Suddivisione in quattro fasce delle squadre partecipanti al mondiale

Per la prima fase del mondiale, le squadre vengono suddivise in gironi tramite sorteggio. I gironi sono composti da quattro squadre ciascuno ed ogni girone contiene esattamente una squadra per fascia, scelta casualmente senza reinserimento. Un possibile risultato del sorteggio è riportato di seguito.

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Canada	Mexico	Brazil	USA	Belgium	Argentina	Morocco	Spain	England	Portugal	Neth.	France
Ecuador	Iran	Senegal	Turkiye	Japan	Germany	Austria	Swiss.	KoreaRep.	Croatia	Uruguay	Colombia
Algeria	Czechia	Egypt	Scotland	CIV	Australia	Norway	Panama	DRCongo	Tunisia	Sweden	Paraguay
SaudiArabia	Ghana	Curacao	Uzbek.	Iraq	Qatar	CaboVerde	NewZealand	Jordan	S.Africa	Haiti	Bos.Herz.

Table 2: Suddivisione in dodici gironi delle squadre partecipanti al mondiale

Nella fase a gironi, ogni squadra gioca contro ciascuna delle altre tre squadre del proprio girone.

1. Definire la funzione `leggi_squadre`

   - **Parametri di ingresso**:
     - `percorso_file`: stringa
   - **Restituisce**: una lista di tuple, ciascuna costituita da una stringa e da un float.
   - **Descrizione**: la funzione apre il file il cui percorso è fornito nel parametro di ingresso e costruisce per ciascuna riga una tupla contenente il primo valore e il secondo valore della riga. Restituisce la lista di tali tuple ordinate in ordine decrescente rispetto al secondo elemento.
   - **Esempio**: se la funzione viene chiamata con il percorso del file `qualificate.txt` come argomento, essa restituisce la seguente lista (troncata per ragioni di spazio):
     `[('France', 1877.32), ('Spain', 1876.40), ..., ('NewZealand', 1281.57)]`

2. Definire la funzione `crea_fasce`

   - **Parametri di ingresso**:
     - `ranking`: lista di tuple, ciascuna costituita da una stringa e da un float.
     - `squadre_ospitanti`: lista di stringhe. Si assuma che ciascuna di tali stringhe compaia negli elementi di `ranking`.
   - **Restituisce**: una lista di liste.
   - **Descrizione**: la funzione considera il primo elemento di ogni tupla della lista `ranking` in ingresso (nome della squadra) e suddivide gli elementi in quattro liste di uguale dimensione. La prima lista viene inizializzata con gli elementi contenuti nella lista `squadre_ospitanti`. Gli altri elementi vengono aggiunti alle liste preservando l'ordine con cui compaiono in `ranking`. La funzione restituisce la lista di liste ottenuta.
   - **Esempio**: la Tabella 1 mostra una rappresentazione formattata dell'output atteso, quando in ingresso viene passato il ranking restituito dalla funzione `leggi_squadre` e `squadre_ospitanti=['Usa', 'Mexico', 'Canada']`. Si noti che non è richiesta la stampa della tabella, ma solo la generazione della struttura dati.

3. Definire la funzione `sorteggio_gironi`

   - **Parametri di ingresso**:
     - `fasce`: lista di liste.
   - **Restituisce**: una lista di liste.
   - **Descrizione**: la funzione costruisce nuove liste di quattro elementi a partire dalle liste ricevute in ingresso. Ciascuna nuova lista viene ottenuta estraendo casualmente un elemento da ciascuna lista contenuta in `fasce`, senza reinserimento. Al termine, la funzione restituisce la lista di liste ottenuta.
   - **Esempio**: la Tabella 2 mostra una rappresentazione formattata di un possibile output atteso, utilizzando le fasce in Tabella 1. Nell'esempio, si ottengono 12 gironi da 4 squadre a partire dalle 4 fasce da 12 squadre. Si noti che non è richiesta la stampa della tabella, ma solo la generazione della struttura dati.

4. Definire la funzione `salva_partite`

   - **Parametri di ingresso**:
     - `gironi`: lista di liste.
   - **Restituisce**: non restituisce alcunché.
   - **Descrizione**: la funzione genera un file di testo (`.txt`) per ciascuna lista contenuta in `gironi`. I file prodotti sono denominati progressivamente usando i numeri naturali a partire da 1. In ciascun file vengono scritte tutte le combinazioni di due elementi della lista corrispondente, una per riga, senza ripetizioni e indipendentemente dall'ordine.
   - **Esempio**: i file prodotti contengono le partite da disputare per ogni girone. Se il primo girone (cioè la prima lista in `gironi`) contiene gli elementi `Canada`, `Ecuador`, `Algeria` e `SaudiArabia`, il file `partite_girone_1.txt` contiene le seguenti righe (l'ordine delle righe non conta).

```
Canada Ecuador
Canada Algeria
Canada SaudiArabia
Ecuador Algeria
Ecuador SaudiArabia
Algeria SaudiArabia
```

Legenda

`s, s1`: stringa `a, b`: numero `i, j, k, n`: intero `x`: elemento generico
`l, l1`: lista `d, d1`: dizionario `r, r1`: set `t, t1`: tupla

Principali funzioni built-in (ordine alfabetico)

`abs(a)`: restituisce il valore assoluto di un numero

`enumerate(iterable)`: restituisce un iteratore i cui elementi sono tuple contenenti il conteggio e il valore ottenuto dall'iterazione su iterable

`input(s)`: scrive il prompt `s` sullo standard output e restituisce stringa acquisita

`isinstance(object, classinfo)`: restituisce True se `object` è del tipo `classinfo` o di una sottoclasse di `classinfo`. Altrimenti, restituisce False

`len(x)`: restituisce la lunghezza (numero di elementi) di `x`. `x` può essere una sequenza (ad es.: string, list, tuple, range) o un contenitore (ad es.: dict, set)

`max(iterable, *, key=None), max(arg1, arg2, *args, key=None)`: restituisce il massimo nell'iterabile, o il massimo tra due o più argomenti. La funzione `key` serve a calcolare il valore da usare per confrontare gli elementi

`min`: analogo a `max`

`print(*x, sep=' ', end='\n')`: stampa oggetti, separati da `sep` e seguiti infine da `end`

`range(j), range(i, j, k)`: restituisce una sequenza immutable per progressione aritmentica

`reversed(seq)`: restituisce un iteratore che percorre `seq` in ordine inverso

`round(a, n)`: arrotonda il valore di `a` all'intero più vicino o ad `n` cifre decimali

`sorted(iterable, key=None, reverse=False)`: restituisce una nuova lista ordinata degli elementi di `iterable`. La funzione `key` serve a calcolare il valore da usare per confrontare gli elementi. `reverse=True` inverte l'ordine

`sum(iterable)`: restituisce la somma dei valori dell'iterabile

`zip(*iterables)`: itera in parallelo su diversi iterabili e restituisce un iteratore costituito da tuple. La i-esima tupla è costituita dall'i-esimo elemento di ciascuno degli iterabili

Indexing e Slicing

`seq[i]`: restituisce l'elemento di indice `i` della sequenza

`seq[i:j]`: restituisce una sottosequenza di `seq` dall'indice `i` all'indice `j` (escluso)

`seq[i:j:k]`: restituisce una sottosequenza di `seq` da indice `i` a indice `j` (escluso) con step `k`

Modulo Random

`choice(seq)`: restituisce un elemento casuale dalla sequenza `seq`

`choices(seq, k=1)`: restituisce una lista di `k` elementi scelti da `seq` con reinserimento

`random()`: restituisce un float casuale in [0,1)

`randint(i, j)`: restituisce un intero casuale tra `i` e `j` (inclusi)

`sample(seq, k)`: restituisce una lista di `k` elementi unici scelti da `seq` senza reinserimento

`shuffle(seq)`: rimescolamento della sequenza `seq` in-place

`uniform(a, b)`: restituisce un float casuale tra `a` e `b` (inclusi)

Principali funzioni del modulo math

`math.cos(a)`, `math.sin(a)`, `math.exp(a)`, `math.log(a)`, `math.log2(a)`, `math.sqrt(a)`

Principali eccezioni

`FileNotFoundError`, `IndexError`, `KeyError`, `ValueError`, `Exception` (eccezione generica)

Stringhe e principali metodi

`str()`, **Literal:** `"abc"` o `'abc'`

Trasformazioni (non in-place): `s.lower()`, `s.upper()`, `s.replace(s1,s2)`, `s.strip()`
Controlli: `s.startswith(s1)`, `s.endswith(s1)`, `s.islower()`, `s.isupper()`, `s.isdigit()`
Da lista a stringa: `s.join(l)`, Da stringa a lista: `s.split(sep)`
Altro: `s.count(s1)`, `s.index(s1)`; Conversione: a intero `int(s)`, a float `float(s)`
Concatenazione: `s1 + s2` Replicazione: `s1 * n` Membership: `s in s1`

Liste e principali metodi

`list()`, **Literal:** `[x1, x2, ...]`

Modifiche (in-place): `l.append(x)`, `l.extend(l1)`, `l.insert(i, x)`, `l.pop(i)`, `l.remove(x)`, `l.reverse()`, `l.sort(key=None, reverse=False)`
Altro: `l.count(x)`, `l.index(x)`
List comprehension: `l = [`*expression* `for` *item* `in` *iterable*`]`
Concatenazione: `l1 + l2` Replicazione: `l1 * n` Membership: `x in l`

Tuple e principali metodi

`tuple()`, **Literal:** `(x1, x2, ...)`

Altro: `t.count(x)`, `t.index(x)`
Concatenazione: `t1 + t2` Replicazione: `t1 * n` Membership: `x in t`

Insiemi e principali metodi

`set()`, **Literal:** `{x1, x2, ...}`

Modifiche (in-place): `r.add(x)`, `r.remove(x)`, `r.discard(x)`, `r.clear()`
Operazioni: `r.difference(r1)` (oppure `r - r1`), `r.symmetric_difference(r1)` (oppure `r^r1`), `r.union(r1)` (oppure `r|r1`), `r.intersection(r1)` (oppure `r&r1`)
Controlli: `r.issubset(r1)`, `r.issuperset(r1)`, `r.isdisjoint(r1)`
Membership: `x in r1`

Dizionari e principali metodi

`dict()`, **Literal:** `{k1:v1, ...}`

Modifiche (in-place): `d1.update(d2)`, `d.pop(key)`
Viste: `d.keys()`, `d.values()`, `d.items()`
Accesso: `d[key]`, `d.get(key, x=None)`
Aggiunta o modifica: `d[key] = val`
Membership: `key in d1`
Dict comprehension: `d = {`*key:val* `for` *item* `in` *iterable*`}`

f-string

`f"abc {`*expression*`:`*format*`}"`

Esempio numeri (cifre decimali): `f"{3.14:.1f}"` → `'3.1'`
Esempio stringa con allineamento (`<`, `>`, `^`): `f"{'ciao':<10}"` → `'ciao␣␣␣␣␣␣'`

```
f"{'Trieste':<10} {10:>3}"   → 'Trieste␣␣␣ ␣10'
f"{'Udine':<10} {7:>3}"      → 'Udine␣␣␣␣␣ ␣␣7'
```

File

Apertura: `open(file, mode, encoding)` apre un file e restituisce un file object `f`
Chiusura: `f.close()`; automatica se il file è aperto con il `with` statement
Lettura: `f.read()` (restituisce una stringa con l'intero contenuto del file), `f.readline()` (restituisce una stringa, fino al prossimo \n), `f.readlines()` (restituisce una lista di stringhe)
Scrittura: `f.write(s)`, `f.writelines(l)`