Soluzioni degli esercizi

Queste soluzioni sono proposte soprattutto per favorire un'acquisizione progressiva delle conoscenze. Bisogna partire dall'assunto che esse *non* siano le uniche o le migliori. Prima di studiarle, ognuno deve cercare in autonomia le *proprie*, che potranno anche essere molto diverse da quelle proposte. Alcune delle soluzioni seguenti potrebbero essere incomplete e presentare solo alcune idee per risolvere gli aspetti più critici del problema.

Esercizi capitolo 11 - Ricorsione

Stringhe palindrome

```
def palindrome(text: str) -> bool:
    if len(text) <= 1:
        return True
    first, last, middle = text[0], text[-1], text[1:-1]
    return first == last and palindrome(middle)
```

▶️ https://fondinfo.github.io/play/?exs/c11_palindrome.py

Una stringa vuota o con un solo carattere è palindroma.
Altrimenti, (1) il primo e l'ultimo carattere devono essere uguali, e (2) la stringa restante dev'essere palindroma.
La stringa senza primo e ultimo carattere si ottiene con una slice: `text[1:-1]`.

Triangolo di Sierpinski

```
def sierpinski(x, y, w, h):
    w2, h2 = w // 2, h // 2
    if w2 < 2 or h2 < 2:
        return
    g2d.draw_rect((x, y), (w2, h2))
    sierpinski(x + w2, y, w2, h2)
    sierpinski(x, y + h2, w2, h2)
    sierpinski(x + w2, y + h2, w2, h2)
```

▶️ https://fondinfo.github.io/play/?exs/c11_sierpinski.py

Un aspetto molto importante per la ricorsione è la definizione dei parametri corretti. In questo caso, per disegnare la figura, occorre conoscere la posizione del rettangolo che la contiene, oltre alla dimensione. Questo rettangolo viene diviso in 2×2 parti. Una viene subito colorata, alle altre viene applicato ricorsivamente il pattern di Sierpinski. Aggiungiamo ora un limite alla ricorsione, passando un parametro in più che funge da contatore decrescente.

```
def sierpinski(x, y, w, h, level):
    w2, h2 = w // 2, h // 2
    if w2 < 2 or h2 < 2 or level == 0:
        return
    g2d.draw_rect((x, y), (w2, h2))
    sierpinski(x + w2, y, w2, h2, level - 1)
    sierpinski(x, y + h2, w2, h2, level - 1)
    sierpinski(x + w2, y + h2, w2, h2, level - 1)
```

Massimo Comun Divisore

```
def gcd(a: int, b: int) -> int:
    if b == 0:
        return a
    return gcd(b, a % b)
```

▶️ https://fondinfo.github.io/play/?exs/c11_gcd.py

La definizione data da Euclide è già ricorsiva. Esiste il caso base. Nel caso generale, è richiesta la ricorsione. Si applica la funzione a problemi via via più semplici, fino ad arrivare al caso base.

Generazione di password

```
def words(alphabet: str, n: int) -> [str]:
    if n == 0:
        return ['']

    ws = words(alphabet, n - 1)
    #return [symbol + word for symbol in alphabet for word in words]

    result = []
    for symbol in alphabet:
        for word in ws:
            result.append(symbol + word)
    return result
```

▶️ https://fondinfo.github.io/play/?exs/c11_words.py

C'è una sola stringa di lunghezza 0. Attenzione a restituire sempre una lista.

Per `n>0`, generiamo tutte le parole di lunghezza `n-1`. A ciascuna parola aggiungiamo come prefisso ciascuna lettera dell'alfabeto.

Anagrammi

```
def anagrams(text: str) -> [str]:
    if len(text) == 0:
        return ['']

    result = []
    for i, char in enumerate(text):
        rest = text[:i] + text[i + 1:]
        for partial in anagrams(rest):
            result.append(char + partial)
    return result
```

▶️ https://fondinfo.github.io/play/?exs/c11_anagrams.py

La lista di anagrammi di una stringa vuota contiene un solo elemento. Attenzione però a restituire sempre una lista.
Se ci sono caratteri, ne estraiamo uno: `char`. Generiamo tutti gli anagrammi con i caratteri restanti, aggiungendo sempre `char` come prefisso.

Bisezione

```
def find_zero(f, low: float, high: float, err: float) -> float:
    x = (low + high) / 2
    y = f(x)
    if abs(y) > err:
        if y * f(low) < 0:
            x = find_zero(f, low, x, err)
        else:
            x = find_zero(f, x, high, err)
    return x
```

▶️ https://fondinfo.github.io/play/?exs/c11_bisection.py
Si divide l'intervallo a metà. Se `f` cambia segno nella prima metà, si ripete il procedimento in questa parte, altrimenti nell'altra.

Torre di Hanoi

```
def print_towers(towers: list):  # ...

def move_towers(towers: list, n: int, src: int, tmp: int, dst: int):
    # if there are discs above, move n-1 away
    if n > 1:
        move_towers(towers, n - 1, src, dst, tmp);

    # now move the largest disc (of n) to its dest
    top_disc = towers[src].pop()
```

```
    towers[dst].append(top_disc)
    print_towers(towers)

    # if there were discs above, move those on top
    if n > 1:
        move_towers(towers, n - 1, tmp, src, dst)

def main():
    discs = int(input("Discs? "))
    towers = [[], [], []]
    for d in reversed(range(discs)):
        towers[0].append(d + 1)
    print_towers(towers)

    # move all discs from pole 0 to pole 2
    move_towers(towers, discs, 0, 1, 2)
```

▶️ https://fondinfo.github.io/play/?exs/c11_hanoi.py

1. Si spostano (ricorsivamente) `n-1` dischi sul paletto di appoggio.
2. Il disco rimanente viene spostato sul paletto di destinazione.
3. Si spostano (ricorsivamente) gli `n-1` dischi dal paletto di appoggio a quello di destinazione.