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.

In queste proposte noterete che i nomi delle variabili, i commenti ecc. sono in inglese. Un suggerimento è quello di provare a operare sul codice per esempio "*traducendolo*" in italiano in modo da riflettere sulla sua logica e il suo contenuto.

Esercizi capitolo 4 - Funzioni

Funzione, Fahrenheit

```
def cels_to_fahr(cels: float) -> float:
    fahr = cels * 1.8 + 32
    return fahr

def main():
    c = float(input("Celsius? "))
    f = cels_to_fahr(c)
    print(f)

main()
```

▶️ https://fondinfo.github.io/play/?exs/c04_fahrcels.py

Area di un'ellisse

```
from math import pi

def ellipse_area(a: float, b: float) -> float:
    return pi * a * b

def main():
    a0 = float(input("a? "))
    b0 = float(input("b? "))
    area = ellipse_area(a0, b0)
    print(area)

main()
```

▶️ https://fondinfo.github.io/play/?exs/c04_ellipse.py

Nelle due funzioni si sono usati nomi diversi per evidenziare che le variabili sono distinte. Anche se i nomi fossero stati uguali, le variabili sarebbero rimaste distinte, perché definite in spazi di nomi diversi.

Perimetro di un triangolo

```
def triangle_perimeter(a: float, b: float, c: float) -> float:
    if a > b + c or b > a + c or c > a + b:
        raise ValueError("Not a triangle")
    return a + b + c

def main():
    again = True
    while again:
        a = float(input("a? "))
        b = float(input("b? "))
        c = float(input("c? "))
        try:
            print(triangle_perimeter(a, b, c))
        except ValueError as e:
            print(e)

        again = input("Continue [Y/N]? ") in "Yy"

if __name__ == "__main__":
    main()
```

▶️ https://fondinfo.github.io/play/?exs/c04_triangle.py

Gruppi di lettere

```
def count_am_nz(text: str) -> tuple[int, int]:
    count_am, count_nz = 0, 0
    for c in text.lower():
        if "a" <= c <= "m":
            count_am += 1
        elif "n" <= c <= "z":
            count_nz += 1
    return count_am, count_nz

def main():
    t = input("Text? ")
    c1, c2 = count_am_nz(t)
    print(c1, c2)

main()
```

▶️ https://fondinfo.github.io/play/?exs/c04_letters.py

Parole di tre lettere

```
def words3(alphabet: str) -> list[str]:
    # …
    result = []
    for c1 in alphabet:
        for c2 in alphabet:
            for c3 in alphabet:
                word = c1 + c2 + c3
                result.append(word)
    return result
```

▶️ https://fondinfo.github.io/play/?exs/c04_words3.py

Possiamo usare tre cicli `for` per ottenere tutte le combinazioni di tre cifre, che leghiamo assieme con una concatenazione. Raccogliamo tutte queste stringhe generate in una lista, che costituisce il risultato richiesto.

Quadrato perfetto

```
def perf_square(n: int) -> tuple[bool, int]:
    i = 1
    while i * i < n:
        i += 1

    if i * i == n:
        return (True, i)
    return (False, 0)

def main():
    n = int(input("n? "))
    perf, root = perf_square(n)

    if perf:
        print("Perfect square of", root)
    else:
        print("Not a perfect square")

if __name__ == "__main__":
    main()
```

https://fondinfo.github.io/play/?exs/c04_perfect.py

Divisori comuni

```
def common_divisors(num1, num2):
    divisors = []
    # Find the smaller number between num1 and num2
    smaller = min(num1, num2)
    # Iterate from 1 to the smaller number
    for i in range(1, smaller + 1):
        # If both numbers are divisible by i ...
        if num1 % i == 0 and num2 % i == 0:
            divisors.append(i)
    return divisors

# Test the function
num1 = 12
num2 = 18
print("Common divisors of", num1, "and", num2, ":")
print(common_divisors(num1, num2))
```

▶️ https://fondinfo.github.io/play/?exs/c04_divisors.py

Disegno di un poligono

```
def draw_polygon(n: int, center: g2d.Point, radius: float):
    angle = 2 * math.pi / n
    for i in range(n):
        pt1 = move_around(center, radius, i * angle)
        pt2 = move_around(center, radius, (i + 1) * angle)
        g2d.draw_line(pt1, pt2)
```

▶️ https://fondinfo.github.io/play/?exs/c04_polygon.py

Ogni punto ottenuto, calcolato per un certo i, è unito al punto successivo, calcolato per $i + 1$.

Orologio classico

```
def draw_watch(center: g2d.Point, radius: int):
    for i in range(60):  # 60 minutes
        radius2 = radius * 0.95  # internal radius is 5% smaller
        if i % 5 == 0:
            radius2 = radius * 0.80  # or 20% smaller, each 5 minutes
        angle = radians(i * 360 / 60)  # 6° rotation for each minute
        pt1 = move_around(center, radius, angle)   # external point
        pt2 = move_around(center, radius2, angle)  # internal point
        g2d.draw_line(pt1, pt2)
```

▶️ https://fondinfo.github.io/play/?exs/c04_watch.py

Conviene ragionare in coordinate polari: raggio e angolo. L'angolo è proporzionale a i. Usiamo uno stesso angolo ma due distanze diverse dal centro, per individuare i due estremi del segmento da disegnare.