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 6 - Oggetti

Classe per ellisse

```
class Ellipse:
    def __init__(self, a0: float, b0: float):
        self._a = a0
        self._b = b0

    def area(self) -> float:
        return pi * self._a * self._b

    def focus(self) -> float:
        return sqrt(abs(self._a ** 2 - self._b ** 2))

def main():
    val_a = float(input("a? "))
    val_b = float(input("b? "))
    e = Ellipse(val_a, val_b)
    a = e.area()
    f = e.focus()
    print(a, f)
```

▶️ https://fondinfo.github.io/play/?exs/c06_ellipse.py

Gli unici campi necessari sono solo quelli che contengono la misura dei semiassi. I metodi hanno sempre come primo e unico parametro `self`. Associati a `self` ci sono anche i campi e quindi anche i semiassi. Perciò, i metodi non necessitano di altri parametri.

Modello quadratico

▶️ https://fondinfo.github.io/play/?exs/c06_???.py

Modello a due variabili

▶️ https://fondinfo.github.io/play/?exs/c06_???.py

Animazione di un veicolo

```
ARENA_W, ARENA_H = 480, 360

class Vehicle:
    def __init__(self, pos: (int, int), dx: int):
        self._x, self._y = pos
        self._dx = dx
        self._xmin, self._xmax = -100, ARENA_W + 100

    def move(self):
        if self._x + self._dx < self._xmin:
            self._x += self._xmax - self._xmin
        if self._x + self._dx > self._xmax:
            self._x -= self._xmax - self._xmin
        self._x += self._dx

    def pos(self):
        return self._x, self._y

    def size(self):
        return 20, 20


def tick():
    g2d.clear_canvas()
    g2d.draw_rect(v1.pos(), v1.size())
    g2d.draw_rect(v2.pos(), v2.size())
    v1.move()
    v2.move()

def main():
    global g2d, v1, v2
    import g2d  # Vehicle does not depend on g2d

    v1 = Vehicle((100, 50), 4)
    v2 = Vehicle((200, 80), -4)
    g2d.init_canvas((ARENA_W, ARENA_H))
    g2d.main_loop(tick)
```

▶️ https://fondinfo.github.io/play/?exs/c06_vehicle.py

Quando il veicolo arriva al suo limite sinistro aggiungiamo `xmax - xmin` in modo che si sposti nella posizione del limite destro. Quando arriva al limite destro, sottraiamo la stessa quantità in modo che si sposti nella posizione del limite sinistro.

Animazione di un alieno

```
W, H = 480, 360

class Alien:
    def __init__(self, pos: (int, int)):
        self._x, self._y = pos
        # TODO: give each alien its own moving space, e.g. 150px
        # self._xmin, self._xmax = ...
        self._dx, self._dy = 5, 5

    def move(self):
        # TODO: use the alien's own limits
        if 0 <= self._x + self._dx <= W - 20:
            self._x += self._dx
        else:
            self._dx = -self._dx
            self._y += self._dy
```

▶️ https://fondinfo.github.io/play/?exs/c06_aliens.py

Quando arriva ai suoi limiti destro e sinistro, l'alieno rimbalza come la pallina. Però, solo nel frame del rimbalzo, anziché muoversi in orizzontale, si muove in verticale.

Ogni alieno ha 150 pixel di movimento. Ma ognuno li misura a partire dalla sua posizione iniziale. Così si muovono tutti assieme, come in un *ballo di gruppo*.

Fantasma con conteggio

```
class Ghost:
    def __init__(self):
        self._x, self._y = ARENA_W // 2, ARENA_H // 2
        self._dx, self._dy = 0, 0
        self._count = 0

    def move(self):
        if self._count > 0:
            self._x = (self._x + self._dx) % ARENA_W
            self._y = (self._y + self._dy) % ARENA_H
            self._count -= 1
        elif randrange(100) == 0:
            self._count = 10
            self._dx = choice([-4, 0, 4])
            self._dy = choice([-4, 0, 4])

    def sprite(self) -> tuple[int, int]:
        if self._count > 0:
```

```
            return 20, 20  # transparent, while moving
        return 20, 0  # visible
#…
```

▶️ https://fondinfo.github.io/play/?exs/c06_nframes.py

Il contatore è inizializzato a zero nel costruttore. Il conto alla rovescia viene avviato all'inizio del movimento. Dopodiché, a ogni frame il contatore viene decrementato fino a raggiungere il valore zero.

Spirale a oggetti

```
class Spiral:
    def __init__(self, center):
        self._center = center
        self._n, self._i = 256, 0
        self._v = 4 * math.pi / self._n  # angular velocity

    def move(self):
        self._i = (self._i + 1) % self._n

    def pos(self):
        i, v, center = self._i, self._v, self._center
        return move_around(center, 25 + i * 0.4, i * v)

    def radius(self):
        return self._i * 0.4

    def color(self):
        return (255 - self._i, 0, self._i)
```

▶️ https://fondinfo.github.io/play/?exs/c06_spiral.py

In questo caso, si ragiona in *coordinate polari*: angolo e raggio. Il raggio, cioè la distanza dall'origine, è proprio `self._i`. Questa distanza è incrementata di 1 a ogni frame. La dimensione e il colore del cerchio sono proporzionali a `self._i`.

Free climbing

```

```

▶️ https://fondinfo.github.io/play/?exs/c06_???.py