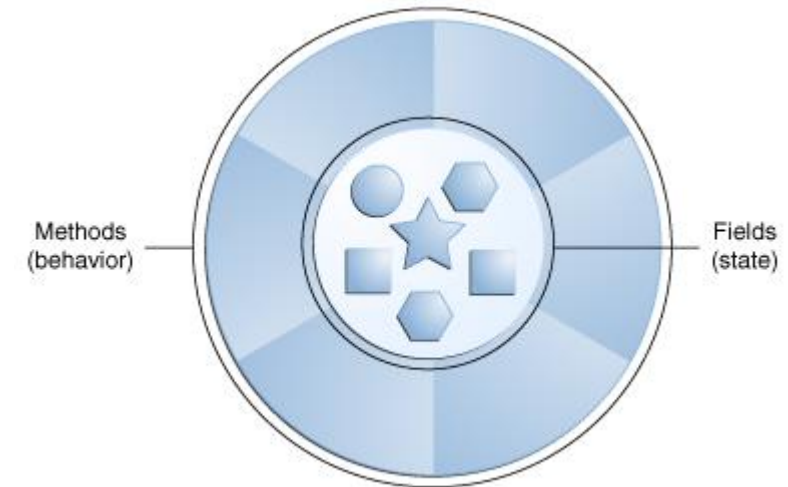




**UNIVERSITÀ  
DI PARMA**

**oggetti**  
**informatica e laboratorio di programmazione**

- analisi della realtà e definizione del ***dominio applicativo***
  - evidenziare informazioni essenziali eliminando quelle non significative per il problema
- un ***oggetto*** rappresenta un oggetto fisico o un concetto del dominio
  - memorizza il suo ***stato*** interno in campi privati (attributi dell'oggetto=
    - concetto di ***incapsulamento*** (black box)
  - offre un insieme di ***servizi***, come ***metodi*** pubblici (comportamenti dell'oggetto)
- realizza un ***tipo di dato astratto***
  - (ADT - *Abstract Data Type*)



- ogni oggetto ha una **classe** di origine (*è istanziato da una classe*)
- la classe definisce la stessa **forma iniziale** (campi e metodi) a tutti i suoi oggetti
- ma ogni oggetto
  - ha la sua **identità**
  - ha uno stato e una locazione in memoria distinti da quelli di altri oggetti
    - sia istanze di classi diverse che della stessa classe



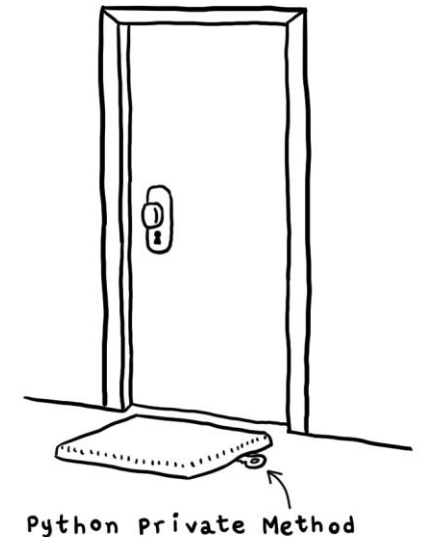
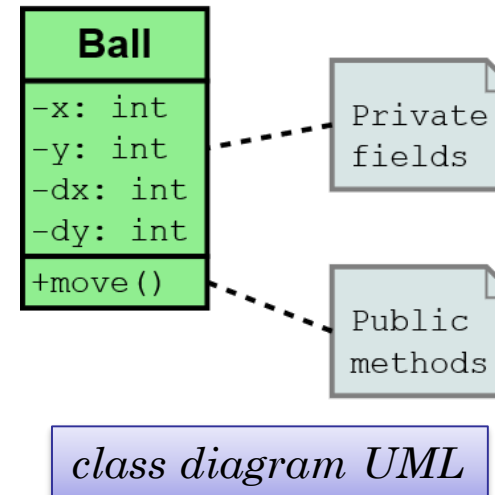
- *incapsulamento* dei dati: convenzione sui nomi
  - prefisso `_` per i nomi dei campi privati

*We're all consenting adults here. (GvR)*

```
class Ball:

    def __init__(self, x: int, y: int):
        self._x = x
        self._y = y
        self._dx = 5
        self._dy = 5
        self._w = 20
        self._h = 20

    # ...
```



Daniel Stori {turnoff.us}

- costruzione di oggetti (*istanziamento*)
- **\_\_init\_\_**: metodo *inizializzatore*
- eseguito *automaticamente* alla creazione di un oggetto
  - instantiation is initialization
- **self**: primo parametro di tutti i metodi
  - non bisogna passare un valore esplicito
  - rappresenta l'oggetto di cui si chiama il metodo
  - permette ai metodi di accedere ai campi



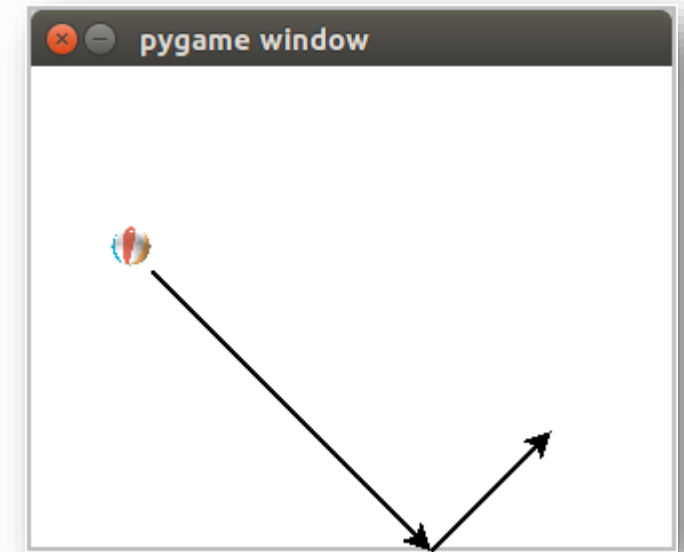
```
ball = Ball(40, 80)  # Allocation and initialization
```

- espongono *servizi* ad altri oggetti

```
ARENA_W, ARENA_H = 320, 240

class Ball:
    # ...
    def move(self):
        if not (0 <= self._x + self._dx <= ARENA_W - self._w):
            self._dx = -self._dx
        if not (0 <= self._y + self._dy <= ARENA_H - self._h):
            self._dy = -self._dy
        self._x += self._dx
        self._y += self._dy

    def position(self) -> (int, int, int, int):
        return self._x, self._y, self._w, self._h
```



```
from cl_ball import Ball # Ball is defined in cl_ball.py

# Create two objects, instances of the Ball class
b1 = Ball(40, 80)        # allocation and initialization
b2 = Ball(80, 40)

for i in range(25):
    print('Ball 1 @', b1.position())
    print('Ball 2 @', b2.position())
    b1.move()
    b2.move()
```

- il primo parametro di ogni metodo si chiama ***self*** (*per convenzione*)
- self rappresenta l'oggetto di cui viene invocato il metodo
- esempio:

`b1 = Ball(40, 80)` *equivale a* `Ball.__init__(b1, 40, 80)`  
`b1.move()` *equivale a* `Ball.move(b1)`

*meglio usare la prima notazione, che evidenzia l'oggetto anzichè la classe!*



```
import g2d
from cl_ball import Ball, ARENA_W, ARENA_H

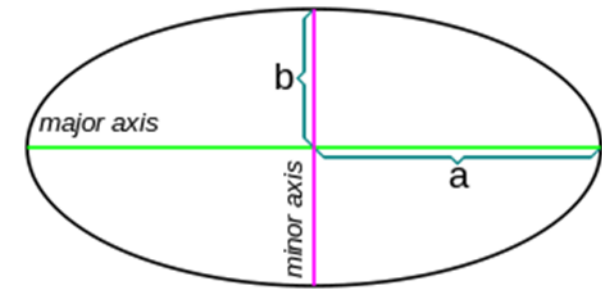
def update():
    g2d.fill_canvas((255, 255, 255)) # BG
    b1.move()
    b2.move()
    g2d.draw_rect((127, 127, 127), b1.position()) # FG
    g2d.draw_rect((127, 127, 127), b2.position()) # FG

b1 = Ball(40, 80)
b2 = Ball(80, 40)
g2d.init_canvas((ARENA_W, ARENA_H))
g2d.main_loop(update, 1000 // 30) # Millis
```

- ***campi***: memorizzano i ***dati caratteristici*** di una istanza
  - ogni pallina ha la sua posizione (x, y) e la sua direzione (dx, dy)
- ***parametri***: ***passano*** altri ***valori*** ad un metodo
  - se alcuni dati necessari non sono nei campi
- ***variabili locali***: memorizzano ***risultati parziali***
  - generati durante l'elaborazione del metodo
  - nomi ***cancellati*** dopo l'uscita dal metodo
- ***variabili globali***: definite ***fuori*** da tutte le funzioni
  - usare sono se strettamente necessario
  - meglio avere qualche parametro in più, per le funzioni

## 4.1 classe ellisse

- definire una **classe** che modella un'ellisse
- **campi privati** (parametri del costruttore)
  - semiassi:  $a$ ,  $b$
- **metodi pubblici** per ottenere:
  - area:  $\pi \cdot a \cdot b$
  - distanza focale:  $2 \cdot \sqrt{|a^2 - b^2|}$
- nel corpo principale del **programma**
  - creare un oggetto con dati forniti dall'utente
  - visualizzare area e distanza focale dell'ellisse



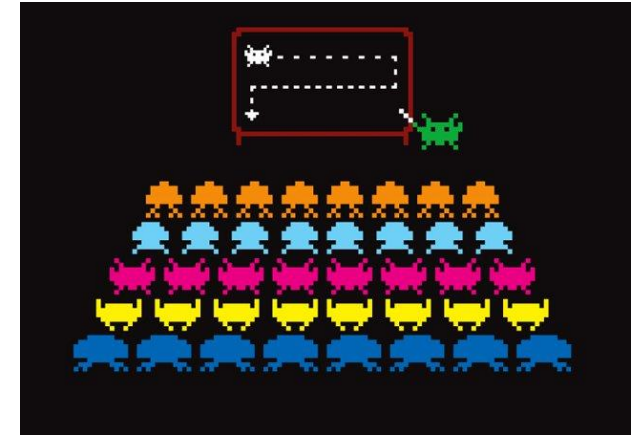
### 4.2 animazione di una pallina

- partire dalla classe **Ball** vista a lezione
- eseguire l'animazione:
  - per ogni frame, chiamare il metodo **move** della pallina
  - rappresentare un rettangolo o un cerchio nella **posizione aggiornata** della pallina
- **modificare** però il metodo move
  - la pallina si sposta sempre di pochi pixel in orizzontale
  - la pallina non si sposta verticalmente
  - se esce dal bordo destro, ricompare al bordo sinistro e viceversa



### 4.3 classe degli alieni

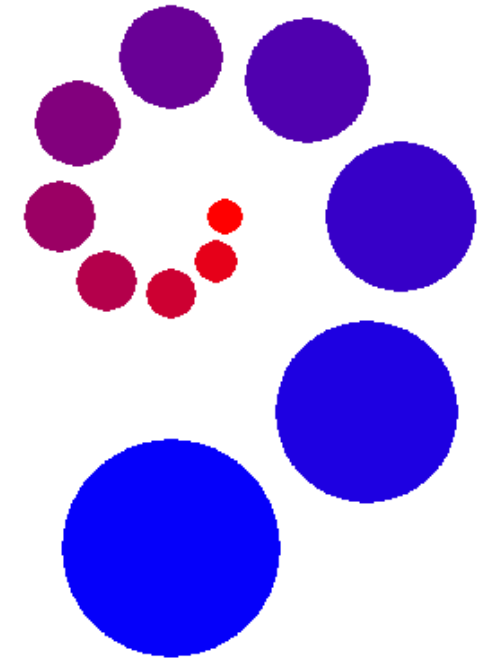
- creare una classe ***Alien*** che contenga i ***dati*** ed il ***comportamento*** dell'alieno
  - campi privati: x, y, dx
  - metodo ***move*** per avanzare
  - metodo ***position*** per ottenere la posizione attuale
- istanziare un ***oggetto*** Alien e farlo muovere sullo schermo
  - chiamare il metodo move ad ogni ciclo
  - visualizzare un rettangolo nella posizione corrispondente



*definire nella classe delle opportune costanti*

### 4.4 spirale

- mostrare l'animazione di un cerchio lungo una **spirale**
- ruotare attorno ad un **centro fisso** ( $x_c, y_c$ )
- **aumentare** la **distanza**  $r$  dal **centro** ad ogni passo
- cancellare lo sfondo ad ogni passo
- disegnare un cerchio sempre **più grande**
- dopo  **$n$**  passi, ricominciare da capo



[http://www.ce.unipr.it/brython/?p2\\_fun\\_spiral.py](http://www.ce.unipr.it/brython/?p2_fun_spiral.py)

### 4.5 classe spirale

- mostrare l'animazione di un cerchio lungo una spirale
- realizzare una classe per gestire dati e comportamento del cerchio
- implementare il movimento in un metodo *move()*
- campi: *xc*, *yc*, *i*
- *i* conta i passi; se eccede il limite, torna a 0

[http://www.ce.unipr.it/brython/?p2\\_oop\\_spiral.py](http://www.ce.unipr.it/brython/?p2_oop_spiral.py)

