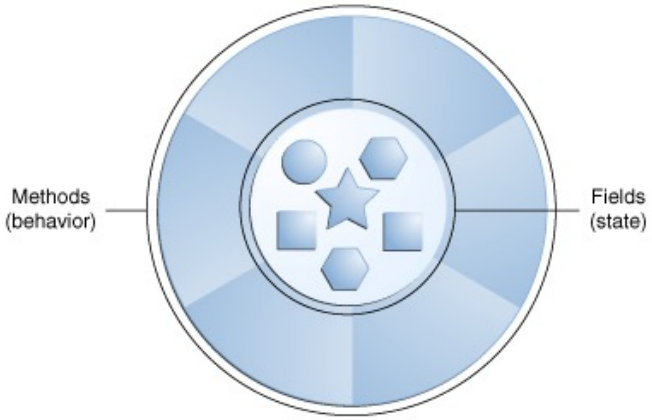
UNIVERSITÀ
DI PARMA

oggetti

oggetto

- analisi della realtà e definizione del ***dominio applicativo***
  - evidenziare informazioni essenziali eliminando quelle non significative per il problema (modello)
- 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*)



classi e oggetti

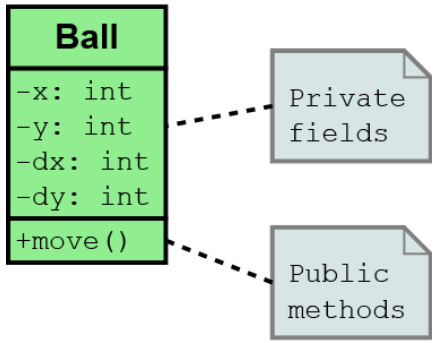
- ogni oggetto ha una ***classe*** di origine (*è istanziato da una classe*)
- la classe definisce la stessa ***forma iniziale*** (campi e metodi) a tutti i suoi oggetti
- ma ogni oggetto
  - ha la sua ***identità***
  - ha uno stato e una locazione in memoria distinti da quelli di altri oggetti
   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

definizione della classe

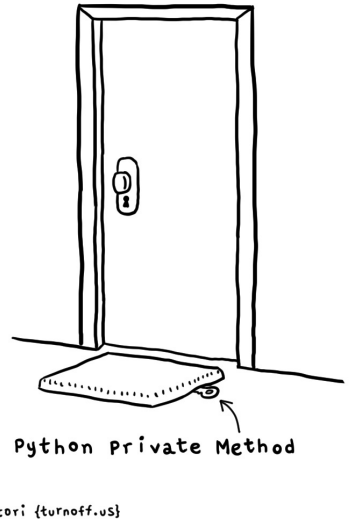
- ***incapsulamento*** dei dati: convenzione sui nomi
  - prefisso _ per i nomi dei campi privati

We're all consenting adults here. (GvR)

```
class Ball:
    # …
    def __init__(self, x0: int, y0: int):
        self._x = x0
        self._y = y0
        self._dx, self._dy = 4, 4
```



class diagram UML



UNIVERSITÀ DI PARMA

costruzione di oggetti

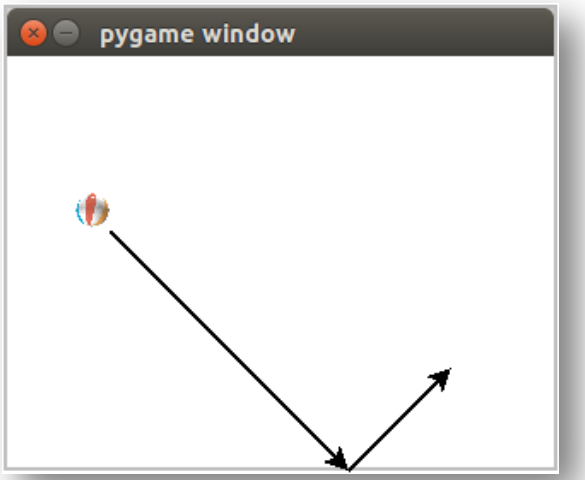
- costruzione di oggetti (***istanziazione***)
- **`__init__`**: metodo ***inizializzatore***
- eseguito ***automaticamente*** alla creazione di un oggetto
  - instantiation is initialization
- **`self`**: primo parametro di tutti i metodi
  - non bisogna passare un valore esplicito
  - rappresenta l'oggetto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

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

metodi

- espongono ***servizi*** ad altri oggetti

```
ARENA_W,ARENA_H, BALL_W,BALL_H = 480,360, 20,20
class Ball: # …
    def move(self):
        if not 0 <= self._x + self._dx <= ARENA_W - BALL_W:
            self._dx = -self._dx
        if not 0 <= self._y + self._dy <= ARENA_H - BALL_H:
            self._dy = -self._dy
        self._x += self._dx
        self._y += self._dy
    def pos(self) -> (int, int): # getter
        return self._x, self._y
```



UNIVERSITÀ DI PARMA

applicazione

```
# Create two objects, instances of Ball class
b1 = Ball(140, 180)
b2 = Ball(180, 140)

b1.move()
b2.move()

print("b1 @", b1.pos())
print("b2 @", b2.pos())
```

self (primo parametro)

- 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 anziché la classe!

UNIVERSITÀ DI PARMA

animazione di due palline

```
b1 = Ball(140, 180)
b2 = Ball(180, 140)


def tick():
    g2d.clear_canvas()
    g2d.draw_image("ball.png", b1.pos())
    g2d.draw_image("ball.png", b2.pos())
    b1.move()
    b2.move()

def main():
    g2d.init_canvas((ARENA_W, ARENA_H))
    g2d.main_loop(tick)
```

https://fondinfo.github.io/play/?c06_ball.py

UNIVERSITÀ DI PARMA

metodi con parametri

- es. più spostamenti consecutivi
- n = parametro del metodo
  - non è caratteristica della pallina
  - è stabilito da chi utilizza l'oggetto
- necessario passare il parametro in fase di chiamata

```
class Ball:
# …
    def multiple_move(self, n: int):
      for i in range(n):
         self.move()

b1 = Ball(40, 40)
b1.multiple_move(3)
b1.multiple_move(2)
```

definizioni

- ***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

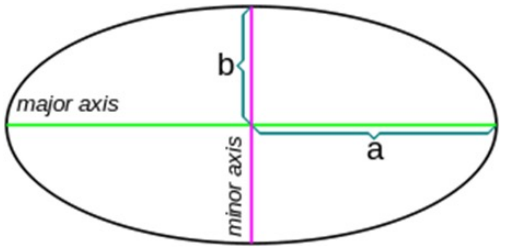
UNIVERSITÀ
DI PARMA

oggetti in python 3

esercizi

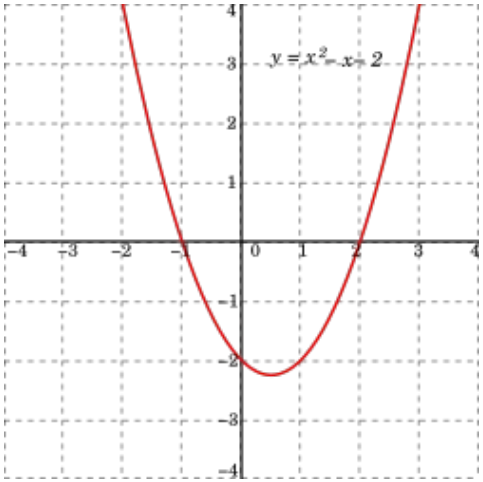
ellisse

- 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



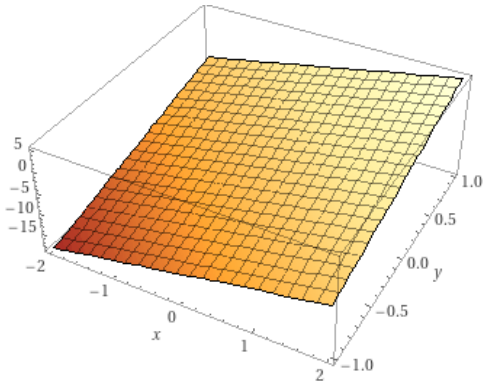
modello quadratico

- creare una classe per rappresentare un modello quadratico a una variabile del tipo: $y=a \cdot x^2+b \cdot x+c$
- inizializzare i coefficienti nel costruttore
- definire poi un metodo ***predict*** che fornisce l'output y del modello in corrispondenza di un certo valore di x
  - x è passato come parametro



modello a due variabili

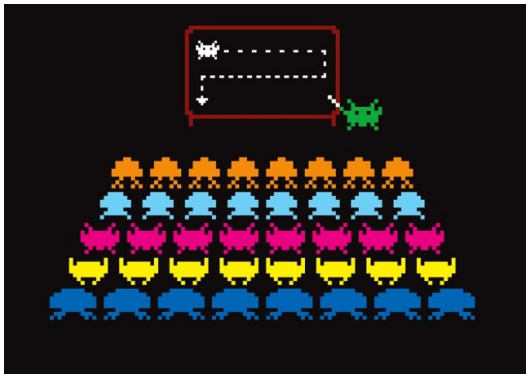
- creare una classe per rappresentare un modello lineare a due variabili del tipo: $z = a \cdot x + b \cdot y + c$
- inizializzare i coefficienti nel costruttore
- definire il metodo ***predict*** che fornisce l'output ***z*** del modello in corrispondenza di un certo valore di ***x*** e un certo valore di **y**
- ***x*** e ***y*** passati come parametri



animazione di una pallina

- partire dalla classe ***Ball***
  - 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

classe di alieni

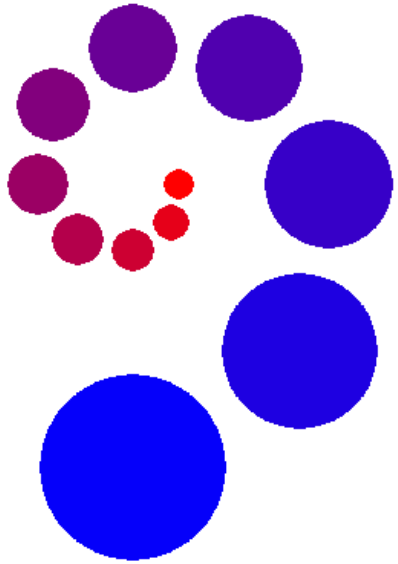


- creare una classe ***Alien***
  che contenga i ***dati*** ed il ***comportamento*** dell'alieno
  - campi privati: x, y, dx
  - metodo ***move*** per avanzare
  - metodo ***position*** per ottenere la posizione attuale
- istanziare un ***oggetto*** Alien e farlo muovere sullo schermo
  - chiamare il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

definire nella classe delle opportune costanti

spirale

- mostrare l'animazione di un cerchio lungo una ***spirale***
- ruotare attorno ad un ***centro fisso*** (xc, yc)
- ***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



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