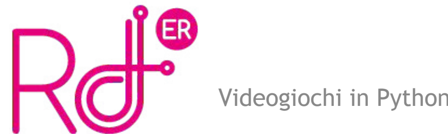
Funzioni e oggetti

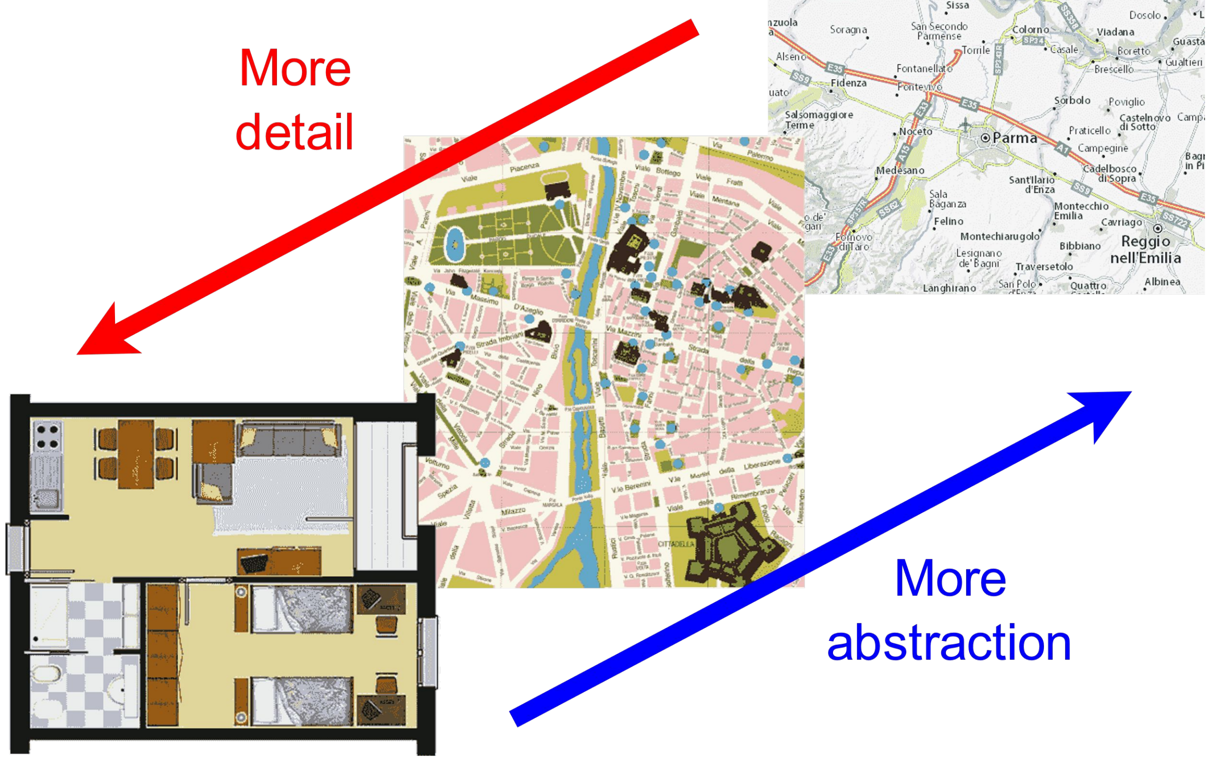
Meccanismi di astrazione

Pensiero astratto

- Astrazione, da “ab trahere” = togliere via ✂️
  - Prescindere da dettagli inessenziali, accidentali
  - Ragionare su concetto o modello, anziché su reale
  - Es. mappe non rappresentano ogni sasso o foglia
- → Generalizzazione
  - Attribuire caratteristiche comuni del concetto a tutte le istanze

- Fondamentale per descrivere e realizzare sistemi software complessi
  - Livelli di astrazione, per incapsulare dettagli 📦
  - Strutture e algoritmi generici, riusabili





The map is not the territory

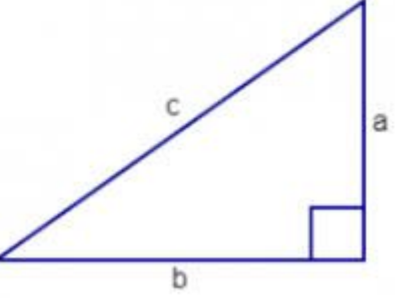
Le funzioni

- Il flusso di esecuzione di un programma può diventare **complesso** ed alcune sue parti **potrebbero dover essere eseguite più volte**

- Ipotizziamo di stare scrivendo un programma che chieda all'utente i lati di 3 diversi rettangoli e ne calcoli poi l'area
    - Non voglio riscrivere tre volte lo stesso codice che calcola l'area per ogni triangolo
    - La formula per calcolarla è sempre la stessa, cambiano solamente gli input

- **Idea**: Definisco un blocco di codice speciale che svolga un generico calcolo dell'area definendolo con dei parametri (generici lati di un triangolo)
- Quando devo calcolare l'area di un triangolo chiederò a Python di sostituire i parametri con i valori effettivi dei lati del mio triangolo e mi limiterò a fare questo ogni volta che devo calcolare una nuova area senza scrivere codice aggiuntivo

- Chiameremo questo blocco speciale di codice una **Funzione**



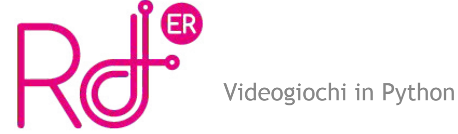
Le funzioni

```python
import math

def calcolo_ipotenusa(cateto_1, cateto_2):
    ipotenusa = math.sqrt( cateto_1 ** 2 + cateto_2 ** 2 )
    return ipotenusa
```

- Operatore, applicato a operandi, per ottenere un risultato
  - `def` per definire una funzione
  - `return` per terminare e restituire un risultato

- Suddivisione di un problema in sotto-problemi
  - Astrazione rispetto all'implementazione
  - Generalizzazione e separazione della soluzione
  - Calcolo basato su parametri di I/O, modello

Le funzioni



Le funzioni

```
def compute_age(birth_day, birth_month,birth_year) :
      age = ...instructions
      return age
```

Scope

Parametri formali
Definiscono il comportamento della funzione (Ingredienti per la torta, generici cateti del triangolo, carote per l'estrattore di succo)

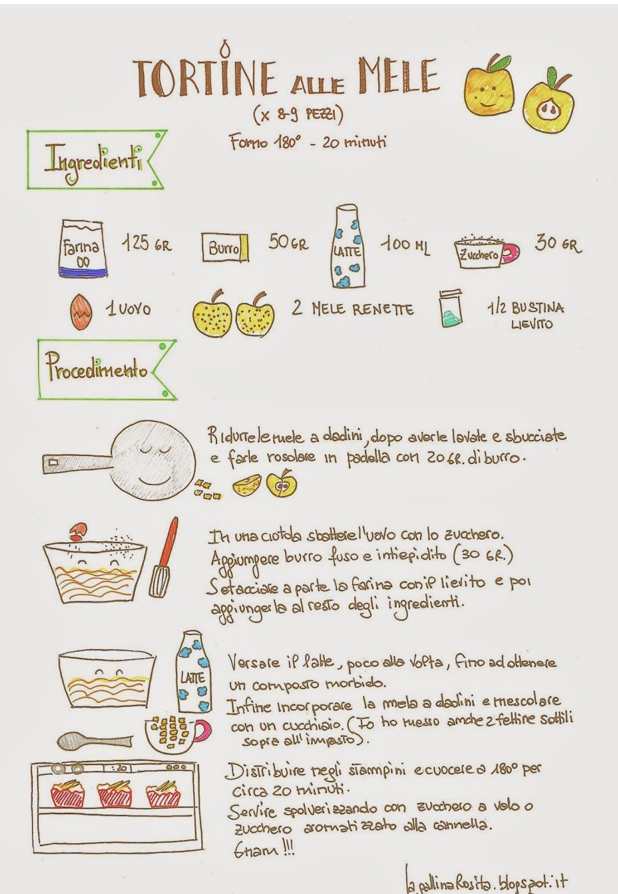
```
marios_age= compute_age(31,7,1981)          #invocation
```

Main corpus

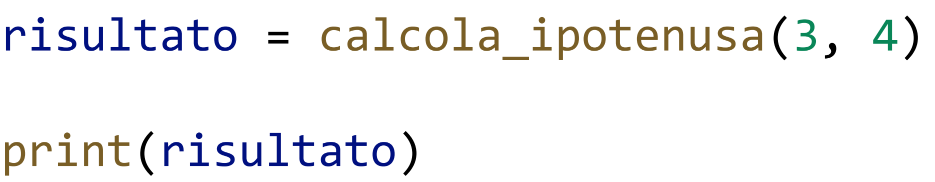
Parametri effettivi
Quando invochiamo una funzione, gli passiamo i dati effettivi su cui deve lavorare:

- La mia farina
- Le mie carote
- Il lato 5 e 7 del triangolo

Le funzioni



Nota bene:



```python
risultato = calcola_ipotenusa(3, 4)

print(risultato)
```

- Chiamata di funzione
  - `def` definisce una funzione, ma non la esegue!
  - Bisogna chiamarla
- Funzione, quando eseguita, crea nuovo spazio di nomi
  - Parametri e variabili hanno ambito locale
  - Non visibili nel resto del programma
  - Nomi uguali, definiti in ambiti diversi, restano distinti
- ⚠️ Ricordarsi di assegnare il risultato ad una variabile
  - Bicchiere per raccogliere la spremuta 🥤

Le funzioni

Syntax

```
def function_name( arg1, arg2... ):
        instructions
        return var      #optionally
.
.
function_name(...) #invokation
```

Py

```
def hypotenuse( leg1, leg2 ):
        hyp = (leg1 ** 2 + leg2 ** 2) ** 0.5

        return hyp      .

hypotenuse(4,5) #invokation
```

- **Attenzione**: Non è un caso che la funzione venga definita nel codice prima di essere invocata, se non la definite prima Python non può sapere quali codici deve andare ad eseguire!
  - Un pò come non potete fare un frullato se prima non acquistate un frullatore

- Dentro ad una funzione può esserci di tutto: costrutti IF, cicli WHILE e FOR

- Le variabili all'interno delle funzioni hanno uno **scope**:
  - Esistono soltanto ed unicamente durante l'esecuzione del codice della funzione!
  - Nel nostro esempio: leg1 e leg2 non possono essere utilizzate fuori dalla funzione perchè semplicemente non esisteranno più!

Le funzioni

```
def compute_age(birth_day, birth_month,birth_year) :
      age = ...instructions
      return age
```

Scope

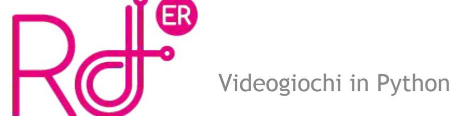
Parametri formali
Definiscono il comportamento della funzione (Ingredienti per la torta, generici cateti del triangolo, carote per l'estrattore di succo)

```
marios_age= compute_age(31,7,1981)        #invocation
```

Main corpus

Parametri effettivi
Quando invochiamo una funzione, gli passiamo i dati effettivi su cui deve lavorare:

- La mia farina
- Le mie carote
- Il lato 5 e 7 del triangolo



Funzione principale

- A volte si preferisce creare una funzione principale, detta `main`
  - In questo modo si limitano le variabili globali
  - Senza `return`, procedura

```
import math

def pitagora(cateto_1, cateto_2):
    ipotenusa = math.sqrt( cateto_1 ** 2 + cateto_2 ** 2 )
    return ipotenusa

def main():
    risultato = pitagora(3, 4)
    print(risultato)

main()
```

Risultato in tupla

```
def div_mod(a, b):
    quoziente = a // b
    resto = a % b
    return (quoziente, resto)

risultato = div_mod(5, 2) # il mio risultato è una tupla
q, r = risultato # 'spacchetto' la tupla
```

Le procedure

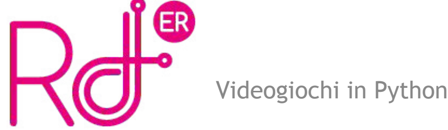
- Abbiamo già visto il main, ma…
- Ci sono altre funzioni senza return
  - solo I/O ed effetti collaterali
  - Astrazione, per riuso e leggibilità
  - Riduce i livelli di annidamento

Animazioni con g2d

```
import g2d

x, y, dx = 50, 50, 5
ARENA_W, ARENA_H = 480, 360

def tick():
    global x, y, dx
    g2d.clear_canvas()                     # Draw background
    g2d.draw_image("ball.png", (x, y))  # Draw foreground
    x += dx                                # Update ball's position

def main():
    g2d.init_canvas((ARENA_W, ARENA_H))
    g2d.main_loop(tick)  # call tick 30 times/second

main()
```

Tick, tastiera e mouse

- `g2d.main_loop`: ciclo di gestione degli eventi
  - Parametro opzionale: funzione che sarà chiamata periodicamente
- `g2d.current_keys`: tutti i tasti attualmente premuti
  - Risultato: tuple[str]
  - Es.: "q", "1", "ArrowLeft", "Enter", "Spacebar", "LeftButton"
- `g2d.mouse_clicked`: controllo se il tasto sx del mouse è stato cliccato
  - Risultato: bool
- `g2d.mouse_pos`: posizione del mouse
  - Risultato: (int, int)

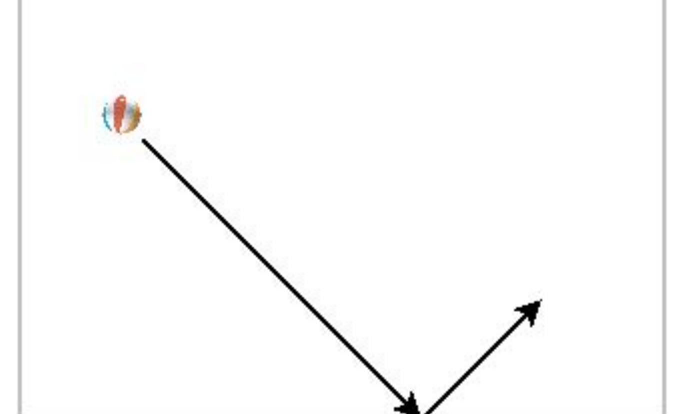


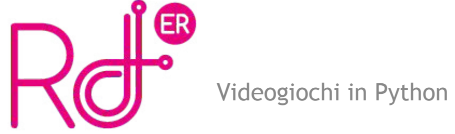
Esercizi :)

Rimbalzi

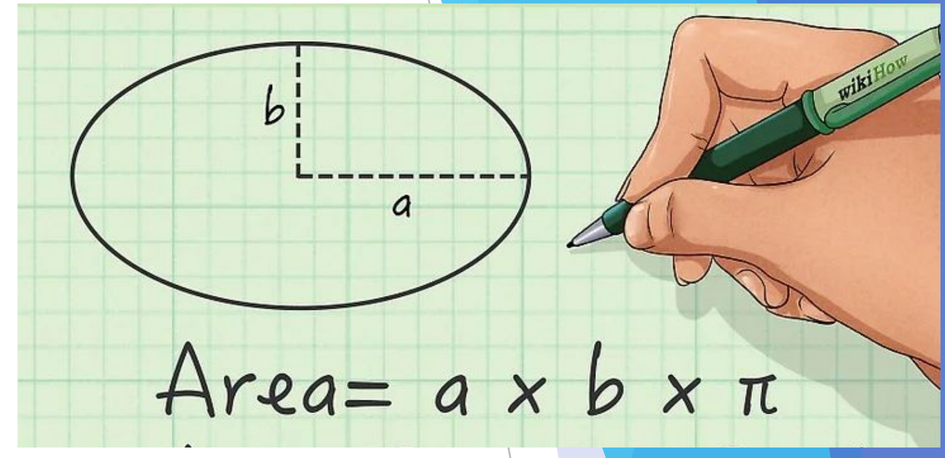
- Scrivere una funzione per animare una pallina, che:
  - Si muova in diagonale
  - Ogni volta che tocca uno spigolo del canvas, rimbalzi e torni indietro!

Provate voi!



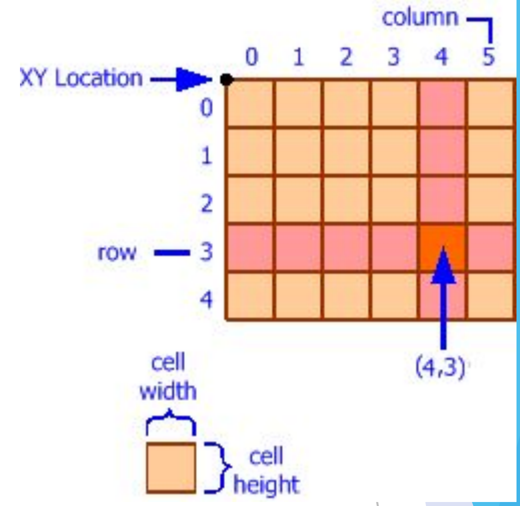


Area di un ellissi



- Definire una *funzione* `ellipse_area` che:
  - Riceve come *parametri* i semiassi di una ellisse: `a, b`
  - Restituisce come risultato l'area dell'ellisse: `π·a·b`
- Definire una *funzione* `main` che:
  - Chiede all'utente due valori
  - Invoca la funzione `ellipse_area` con questi parametri
  - Stampa il risultato ottenuto

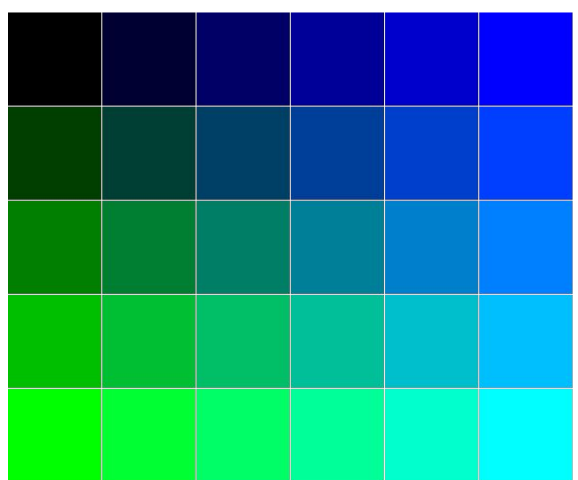
Griglia di colori



- Chiedere all'utente dei valori per `rows` e `cols`
- Mostrare una griglia di rettangoli di dimensione `rows×cols`
- Partire da un rettangolo nero in alto a sinistra
- In orizzontale, aumentare gradatamente la componente di blu
- In verticale, aumentare gradatamente la componente di verde

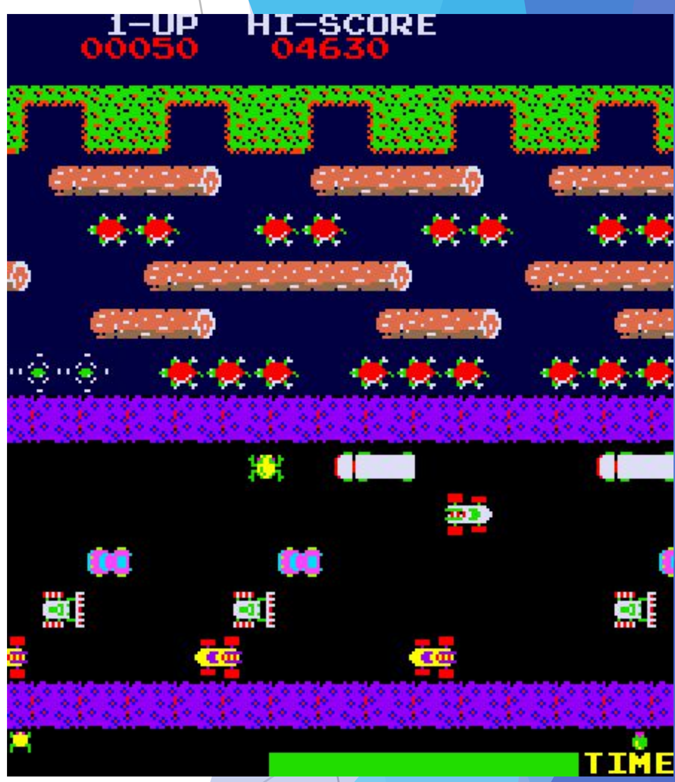
Cominciare a creare una griglia di riquadri tutti neri, con due cicli annidati

Lasciare tra i riquadri un piccolo margine



Movimento orizzontale

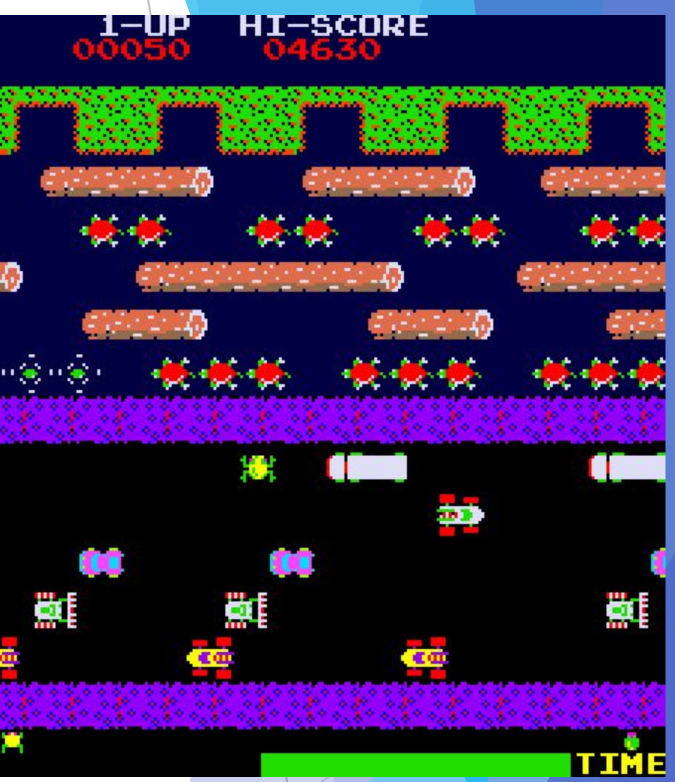
- Mostrare una pallina che si muove in orizzontale
  - Variabile `dx` indica lo spostamento da effettuare ad ogni ciclo
- La pallina riappare dal bordo opposto, dopo un po' di tempo
  - Permettere alla pallina di superare i bordi laterali, p.es. di *100px*
  - Se supera 100px oltre il bordo destro, ricompare a 100px prima del bordo sinistro e viceversa
- Al click del mouse, la pallina inverte la direzione



Movimento per 5 fotogrammi

- Mostrare una pallina che si muove in orizzontale
  - Variabile `dx` indica lo spostamento da effettuare ad ogni ciclo
- La pallina si muove solo dopo il click del mouse
  - Si sposta solo per 5 fotogrammi
  - Dopo si ferma, fino a nuova pressione
- Invertire la direzione ad ogni avvio del movimento

Incrementare (o decrementare) un contatore ad ogni chiamata a `tick`



Videogiochi in Python

Creiamo il nostro primo gioco testuale

La caverna del Drago

Prima di scrivere codice…

- È importante curare bene il design del gioco

- Che tipo di gioco è?
  - C'è una storia? Ci sono dei livelli?
- Come si gioca?
  - Quali sono i comandi a disposizione del giocatore?
  - Quando si vince? Quando si perde?

- Infine, qual è la logica del gioco?

La caverna del drago

- Ad esempio, il nostro potrebbe essere un gioco testuale
  - Come un libro interattivo
  - Il giocatore leggerà sulla shell interattiva la situazione
  - E gli verrà chiesto di scegliere tra alcune possibili mosse
  - La situazione si evolve in base alle risposte del giocatore

- In questo gioco, vogliamo che il giocatore trovi la strada giusta all’interno di una caverna, per trovare il tesoro ed evitare il drago!
  - Il giocatore non ha indizi

La caverna del drago

- Decidiamo noi designer quanti bivi ci sono nella caverna
- Ma la strada giusta non è sempre la stessa!
  - Il tesoro e il drago possono essere posizionati in modo random