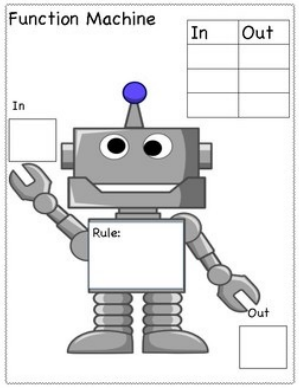
UNIVERSITÀ
DI PARMA

funzioni



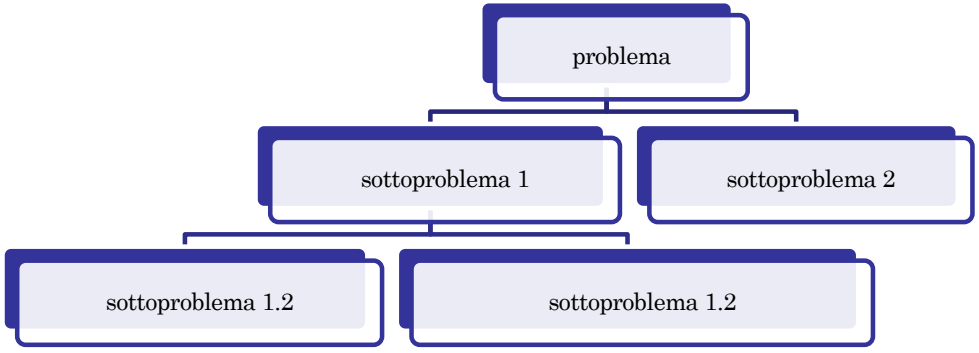
definizione di funzione

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

```
def hypotenuse(a, b):
    c = sqrt(a ** 2 + b ** 2)
    return c
```

funzioni e sottoproblemi

- suddivisione di un problema in sottoproblemi
- un sottoproblema può essere risolto mediante una funzione
- astrazione rispetto all'implementazione
  - per utilizzare una funzione non è necessario conoscere i dettagli della sua implementazione
  - una funzione diventa una istruzione non primitiva attivabile in qualsiasi punto del programma
- importante che la funzioni operi su parametri e restituisca un risultato



UNIVERSITÀ DI PARMA

chiamata di funzione

- **`def`** definisce una funzione, ma non la esegue!
- per far ***eseguire*** una funzione è necessario «***chiamarla***»
  - la funzione, quando viene eseguita, crea nuovo spazio di nomi
  - i parametri e le variabili hanno ambito locale
  - non sono visibili nel resto del programma
  - nomi uguali, definiti in ambiti diversi, restano distinti

```
side1 = float(input("1st side? "))
side2 = float(input("2nd side? "))
side3 = hypotenuse(side1, side2)
print("3rd side:", side3)
```

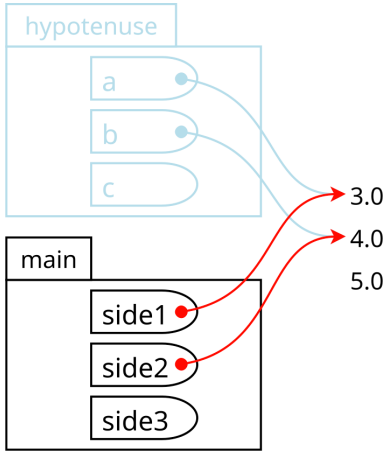
funzione main

- è spesso preferibile creare una ***funzione principale (main)***
- in questo modo si limitano le variabili globali

```
def hypotenuse(a, b):
    c = sqrt(a ** 2 + b ** 2)
    return c

def main():
    side1 = float(input("1st side? "))
    side2 = float(input("2nd side? "))
    side3 = hypotenuse(side1, side2)
    print("3rd side:", side3)

main() # remove, if importing the module elsewhere
```



parametri di funzioni

- la definizione della funzione opera sui ***parametri formali***
- al momento della chiamata si definiscono i ***parametri attuali***
- le variabili definite nella funzione rimangono locali a questa

```
def dummy(f1, f2):
    loc = f1 ** f2
    f1 = f1 * 2
    return loc

a1 = float(input("fist value: "))
a2 = float(input("secondt value: "))
print(dummy(a1,a2))
print(loc)     # NameError: name 'loc' is not defined
print(a1)      # print ???
```

passaggio dei parametri

- ***call-by-object***
- parametri passati «per oggetto»
  - se il parametro è ***non mutabile*** le modifiche non si ripercuotono all'esterno
  - se il parametro è ***mutabile*** (es ***lista)*** le modifiche si ripercuotono

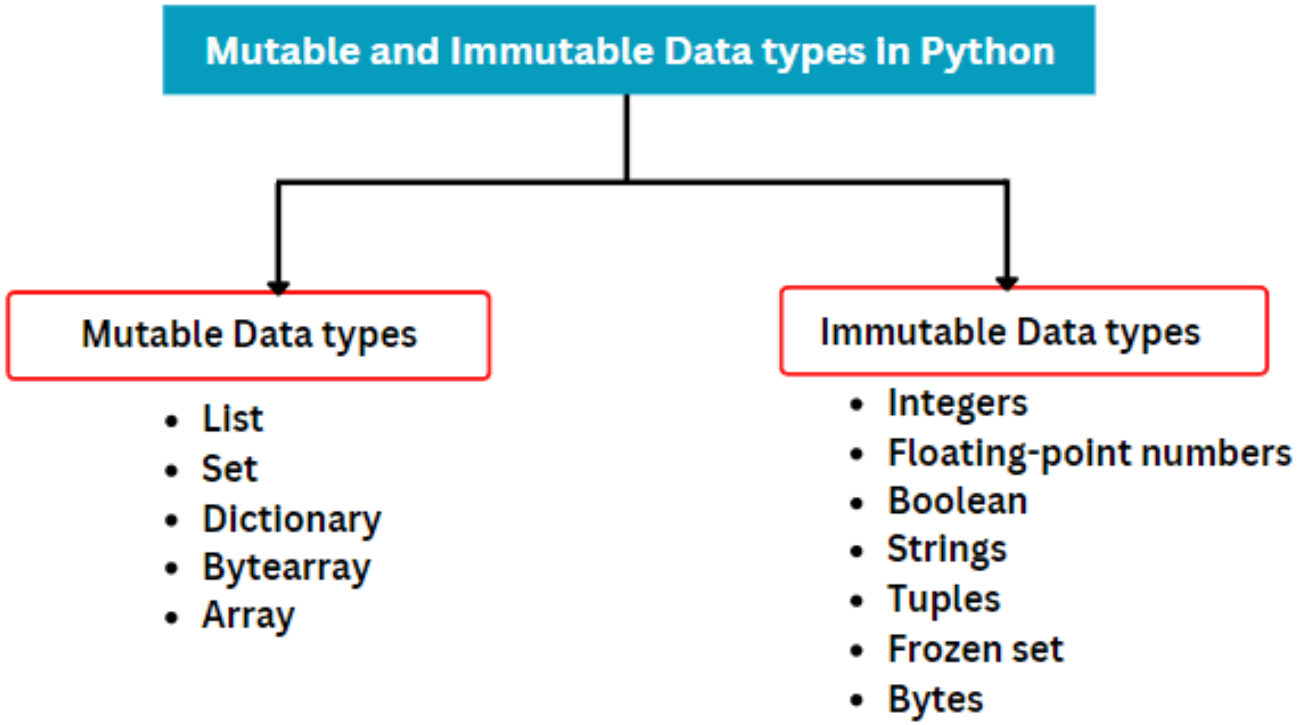
```
def inc(f):
    f = f + 1
    print(f) # 11

a = 10
inc(a)
print(a)     # 10
```

```
def inc(f):
    for i in range(0,len(f)):
        f[i] = f[i] + 1
    print(f) # [3,4,6]

a = [2,3,5]
inc(a)
print(a) # [3,4,6]
```

tipi mutabili e non mutabili



UNIVERSITÀ DI PARMA

restituzione di più valori come tupla

```
def min_max(f):
        '''
        restituisce valore minimo e massimo della lista f
        '''
        minimo = massimo = f[0]
        for i in range(1,len(f)):
                if f[i] < minimo:
                        minimo = f[i]
                if f[i] > massimo:
                        massimo = f[i]
        return minimo, massimo

def main():
        a = [2,13,5,-3,8]
        x , y = min_max(a)
        print("minimo: ",x," massimo: ",y)

main()  ## remove if importing the module elsewhere
```

UNIVERSITÀ DI PARMA

documentazione di funzioni

- ***annotazioni***: utili per documentare il tipo dei parametri e il tipo del valore di ritorno *(ma non c'è verifica!)*
- ***docstring***: descrizione testuale di una funzione
- ***help***: funzione per visualizzare la documentazione

```
def hypotenuse(cathetus1: float, cathetus2: float) -> float:
    '''
    Return the hypotenuse of a right triangle, given both its legs (catheti).
    '''
    return (cathetus1 ** 2 + cathetus2 ** 2) ** 0.5
```

https://mypy.readthedocs.io/en/latest/cheat_sheet_py3.html

UNIVERSITÀ DI PARMA

docstring

- la stringa di documentazione, posta all'inizio di una funzione, ne ***illustra l'interfaccia***
- per convenzione, la ***docstring*** è racchiusa tra triple virgolette, che le consentono di essere divisibile su più righe
- è breve, ma contiene le informazioni essenziali per usare la funzione
  - spiega in modo conciso ***cosa fa*** la funzione (non come lo fa)
  - spiega il significato di ciascun parametro e il suo tipo
- è una parte importante della progettazione dell'interfaccia
  - un'interfaccia deve essere ***semplice*** da spiegare

UNIVERSITÀ DI PARMA

procedura

- funzione ***senza return***
  - non restituisce valori
  - solo I/O ed effetti collaterali
- ***astrazione***, per riuso e leggibilità
- riduce i livelli di annidamento

```
def print_row(y: int, size: int):
    for x in range(1, size + 1):
        val = x * y
        print(f"{val:3}", end=" ")
    print()

def print_table(size: int):
    for y in range(1, size + 1):
        print_row(y, size)

def main():
    print_table(10)
```

UNIVERSITÀ DI PARMA

animazione

- g2d fornisce la possibilità di richiamare dopo un certo intervallo di tempo una funzione
  - ***g2d.main_loop(f,n)***
  - ***f*** è il nome della funzione da richiamare
  - ***n*** è l'intervallo di tempo (n volte al secondo)
- la funzione può generare un'animazione cancellando tutto o una parte del canvas e facendo apparire una nuova immagine
- nell'esempio viene visualizzata una pallina in posizione casuale ogni mezzo secondo

```
import random, g2d
g2d.init_canvas((500, 500))

def visualizza():
    x = random.randrange(500)
    y = random.randrange(500)
    g2d.clear_canvas()                      # Draw background
    g2d.draw_image("ball.png", (x, y))      # Draw foreground

g2d.main_loop(visualizza,2)                 # call visualizza 2 times/second
```

UNIVERSITÀ DI PARMA

variabili locali e globali

- ***variabili globali***: sono definite al di fuori di una funzione
  - sono visibili in tutto il programma, comprese le funzioni (se non vengono ridefinite all'interno)

```
x = 10  # variabile globale

def stampa_x():
    print(x)  # può accedere alla variabile globale

stampa_x()  # Output: 10
```

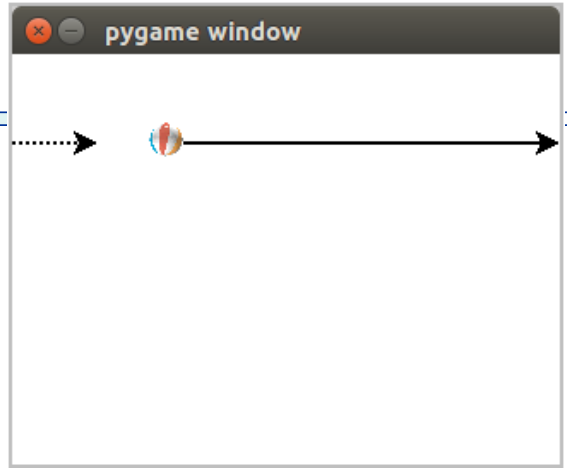
- per fare in modo che una funzione possa ripercuotere all'esterno le modifiche a variabili locali non mutabili è necessario utilizzare all'interno della funzione variabili globali
  - global x

```
x = 5

def raddoppia_x():
    global x  # indica che vogliamo usare la x globale
    x = x * 2

raddoppia_x()
print(x)  # Output: 10
```

animazione




```
import g2d

x, y, dx = 40, 40, 4
ARENA_W, ARENA_H = 480, 360

def tick():
    global x, dx
    g2d.clear_canvas()                      # Draw background
    g2d.draw_image("ball.png", (x, y))      # Draw foreground
    if g2d.mouse_clicked():
        dx = -dx
    if x + dx > ARENA_W or x + dx < 0:
        dx = -dx

    x += dx                                 # Update ball's position

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

UNIVERSITÀ DI PARMA

animazione con tipo mutabile

```
import g2d

data = [10,10,4]              # x pos, y pos, dx
ARENA_W, ARENA_H = 600, 400

def tick():
    x,y,dx = data
    g2d.clear_canvas()                      # Draw background
    g2d.draw_image("ball.png", (x, y))      # Draw foreground
    if g2d.mouse_clicked():
        dx = -dx
    if x + dx > ARENA_W or x + dx < 0:
        dx = -dx
    x += dx                                 # Update ball's position
    data[0]=x
    data[2]=dx

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

UNIVERSITÀ DI PARMA

g2d - eventi

- **g2d.main_loop**
  - **ciclo di gestione degli eventi**
  - parametri: funzione da chiamare ciclicamente, tempo di attesa
- **g2d.key_pressed – g2d.key_released**
  - controllo pressione (rilascio) tasto
  - risultato: bool , parametro: str (nome del tasto)
- **g2d.mouse_position**
  - posizione del mouse
  - risultato coordinate (int, int)
- **g2d.current_keys**
  - tutti i tasti attualmente premuti
  - risultato: sequenza di str – Possibili valori
  - Es.: "q", "1", "ArrowLeft", "Enter", "Spacebar", "LeftButton"
- **g2d.close_canvas**
  - chiude il canvas, termina l'esecuzione

documentazione g2d
https://github.com/fondinfo/fondinfo#g2d

UNIVERSITÀ DI PARMA

funzioni in moduli

- un file Python è un modulo: nome del file, senza .py
- se importato altrove, esecuzione main sotto condizione

```
# file `mymath.py`
def hypotenuse(leg1: float, leg2: float) -> float:
    return sqrt(leg1 ** 2 + leg2 ** 2)

def main():  # Use or test the hypotenuse function
    print(hypotenuse(3, 4))

if __name__ == "__main__":  # file executed directly, or imported?
    main()  # `main` not called, if file imported as module
```

```
>>> import mymath # nothing printed
>>> mymath.hypotenuse(4, 3)
5
```

UNIVERSITÀ DI PARMA

condizioni di errore

- ***precondizioni*** per attivazione non soddisfatte
  - ⇒ errore segnalato con instruzione ***raise***
- es. triangolo errato

```
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

print(triangle_perimeter(4, 2, 1))
```

UNIVERSITÀ DI PARMA

più return in funzione

- una funzione può presentare più istruzioni ***return***
  - l'esecuzione termina quando si incontra ***return*** e il controllo torna al codice chiamante
  - *violazione accettabile della programmazione strutturata*

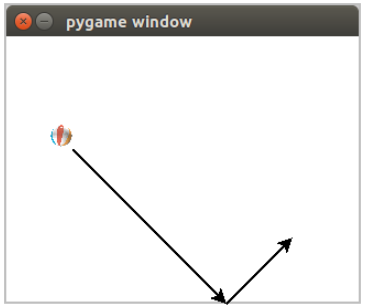
```
def exceeds(data: list[int], limit: int) -> bool:
    for v in data:
        if v > limit:
            return True
    return False
```

- se non si incontra nessuna istruzione return o se return non specifica nessun valore il risultato della funzione è ***none***

rimbalzi

- le funzioni forniscono ***limitata astrazione***
  - incapsulano il comportamento
  - ma espongono i dati

```
def move_ball(x: int, y: int,
              dx: int, dy: int) -> tuple[int, int, int, int]:
    if not 0 <= x + dx <= ARENA_W - BALL_W:
        dx = -dx
    if not 0 <= y + dy <= ARENA_H - BALL_H:
        dy = -dy
    x += dx
    y += dy
    return (x, y, dx, dy)
```



https://fondinfo.github.io/play/?c05_move.py

UNIVERSITÀ DI PARMA

effetti collaterali

- la funzione può modificare oggetti passati come parametri o variabili globali o effettuare operazioni di lettura/scrittura...
  - effetti collaterali annullano la ***trasparenza referenziale***
- impossibile semplificare, sostituendo una chiamata a funzione col suo valore di ritorno (es. presenti operazioni di I/O)
- effetti collaterali rendono la funzione ***non idempotente***
- chiamata più volte, con gli stessi parametri, la funzione può restituire risultati diversi
  - → difficile fare verifiche matematiche

```
z = f(sqrt(2), sqrt(2))
s = sqrt(2)
z = f(s, s)
```

funzioni non idempotenti

- esempio di semplificazione

```
p = f(x) + f(y) * (f(x) - f(x))
p = f(x) + f(y) * (0)
p = f(x) + 0
p = f(x)
```

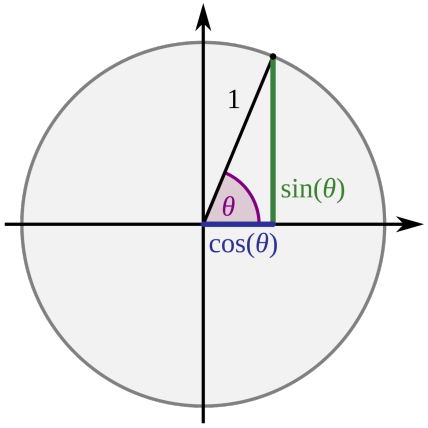
- ma se **f** ha effetti collaterali non è corretto

```
base_value = 0  # global variable

def f(x: int) -> int:
    global base_value
    base_value += 1
    return x + base_value
```

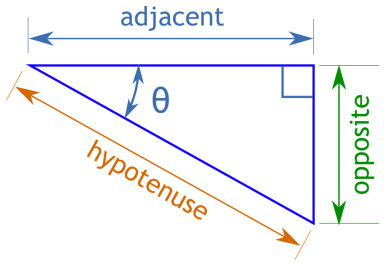
UNIVERSITÀ
DI PARMA

funzioni trigonometriche



coordinate polari

- *cos* e *sin* sono definite come il rapporto tra le lunghezze di uno dei cateti e l'ipotenusa di un triangolo rettangolo
- note l'ipotenusa e un angolo si possono ricavare i cateti
- ***coordinate polari*** di un punto = coppia di valori **(r, θ)**
  - r distanza dall'origine
  - θ angolo rispetto all'asse delle ascisse



- coordinate polari (r,θ) ⇒ coordinate cartesiane (x,y)

$$\begin{cases} x = r \cdot \cos(\theta) \\ y = r \cdot \sin(\theta) \end{cases}$$

- coordinate cartesiane (x,y) ⇒ coordinate polari (r,θ)

$$\begin{cases} r = \sqrt{x^2 + y^2} = hypot(x, y) \\ \theta = \text{atan2}\left(\frac{y}{x}\right) \end{cases}$$

UNIVERSITÀ DI PARMA

esempio disegno di raggi

- spostamento (r,θ) rispetto al punto (x_0,y_0)
  - traslazione
- nel modulo math costanti e funzioni utili
  - sin, cos, radians, pi, hypot, atan2, dist …
- esempio: 4 raggi con angoli ***0,15,30,45*** dal punto ***(x_0,y_0)*** e raggio ***r***

```
def draw_rays(x0: int, y0: int, r: int):
    for angle in [0, 15, 30, 45]:
        x = x0 + r * cos(radians(angle))
        y = y0 + r * sin(radians(angle))
        g2d.draw_line((x0, y0), (x, y))
```

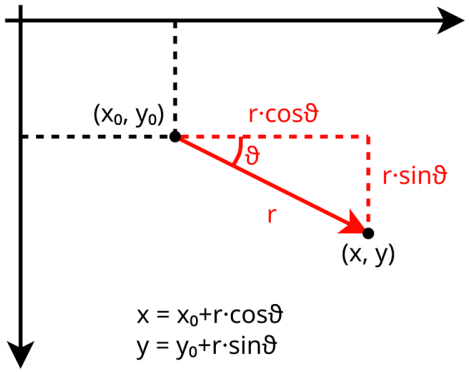
https://fondinfo.github.io/play/?c04_angles.py

funzioni su coordinate polari

```
Point = tuple[float, float]  # Pt in cartesian coords (x, y)
Polar = tuple[float, float]  # Pt in polar coords (r, angle)

def from_polar(plr: Polar) -> Point:
    r, a = plr
    return (r * cos(a), r * sin(a))

def move_around(start: Point, length: float, angle: float) -> Point:
    x0, y0 = start
    dx, dy = from_polar((length, angle))
    return x0 + dx, y0 + dy
```



esercizio: riscrivere la funzione ***draw_rays*** usando ***move_around***

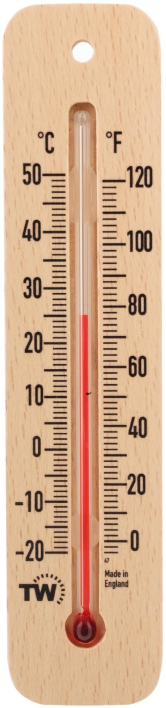
UNIVERSITÀ
DI PARMA

funzioni in python 3

esercizi

fahrenheit

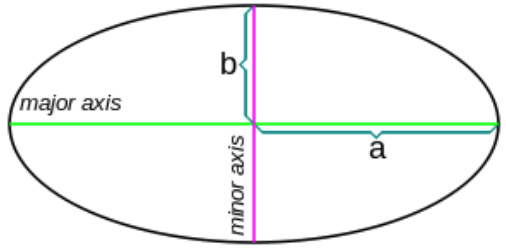
- definire la funzione ***cels_to_fahr***
  - parametro: temperatura Celsius di tipo float
  - risultato: temperatura Fahrenheit di tipo float
- invocare la funzione dalla shell interattiva
- definire poi la funzione ***main***
  - procedura senza parametri e senza risultato
  - chiedere all'utente la temperatura Celsius
  - chiamare ***cels_to_fahr***
  - stampare il risultato



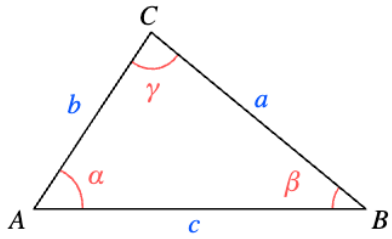
formula: fahr = cels * 1.8 + 32

area ellisse

- definire la funzione ***ellipse_area*** che:
  - riceve come parametri i ***semiassi*** di una ellisse: **a, b**
  - restituisce come risultato l'area dell'ellisse: $\pi \cdot a \cdot b$
- definire la funzione ***main*** che:
  - chiede all'utente due valori
  - invoca la funzione ellipse_area con questi parametri
  - stampa il risultato ottenuto



triangolo - perimetro



- definire la funzione ***triangle_perimeter***
  - riceve come parametri i tre lati di un triangolo ***a, b, c***
  - restituisce come risultato il perimetro del triangolo
- se i tre lati non formano un triangolo genera un ***ValueError***
  - (uno dei lati è maggiore della somma degli altri due)
- definire la funzione ***main***
  - ciclicamente chiede all'utente tre valori
  - mostra il risultato di ***triangle_perimeter*** con questi parametri
  - chiede all'utente se vuole elaborare altri dati o terminare l'esecuzione

gruppi di lettere

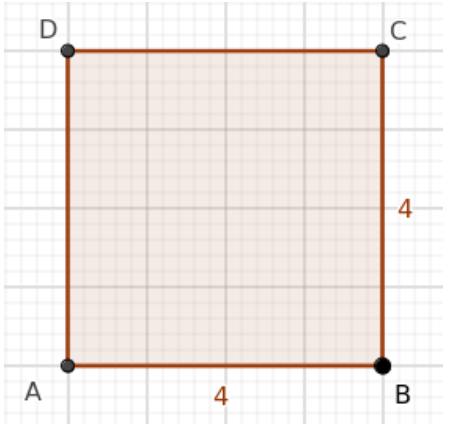
- definire una funzione ***count_groups***
  - parametro testuale (str)
  - risultato due valori:
    - quante lettere del testo sono comprese nel gruppo A-M
    - quante lettere del testo sono comprese nel gruppo N-Z
- la funzione count_groups non distingue fra lettere maiuscole e minuscole
- chiamare la funzione count_groups con un testo fornito dall'utente e mostrare i risultati

parole di 3 lettere

- definire la funzione ***padlock*** con un parametro di tipo str
  - la stringa rappresenta un alfabeto di caratteri validi
  - la funzione genera la lista di tutte le parole di lunghezza esattamente uguale a 3 composte con i soli caratteri dell'alfabeto fornito in input
- se l'alfabeto contiene lettere ripetute ***padlock*** solleva un ***ValueError***

quadrato perfetto

- definire la funzione ***perfect_square***
  - parametro un numero intero
  - verifica se si tratta di un quadrato perfetto
    - la sua radice quadrata è un numero naturale?
  - risultati: booleano, numero
    - true, radice perfetta del numero se esiste
    - false, 0

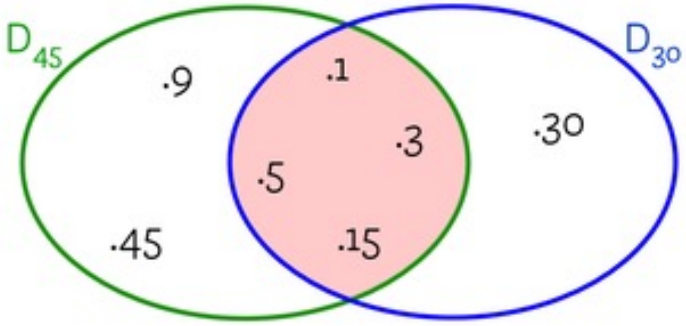


- definire ***main*** per eseguire perfect_square su un numero fornito dall'utente e mostrare il risultato

non usare math.sqrt e simili — risolvere con una iterazione

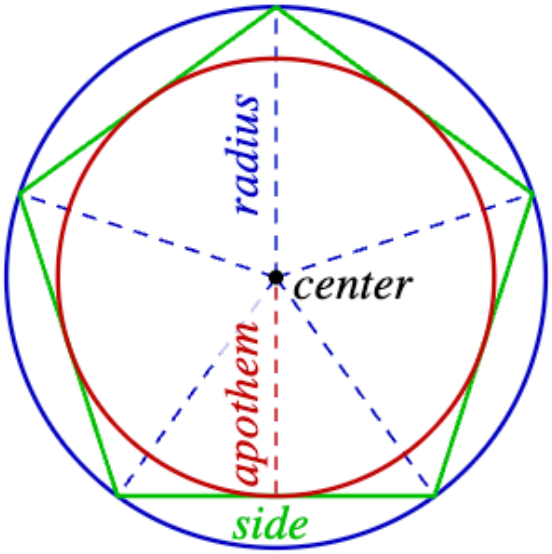
divisori comuni

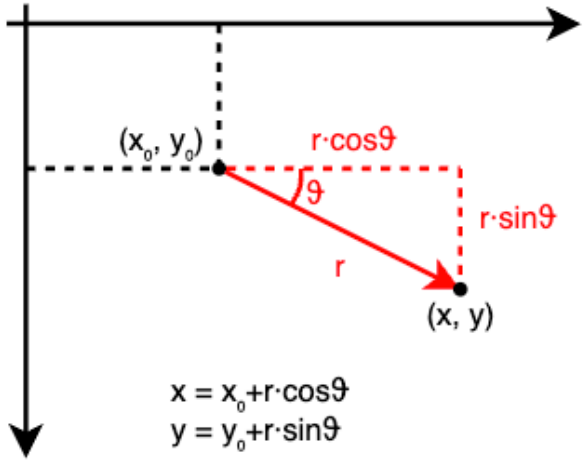
- definire la funzione ***common_divisors***
  - parametri: due numeri naturali
  - restituisce una lista con tutti i divisori comuni ai due numeri
- definire ***main*** per eseguire common_divisors su numeri forniti dall'utente e mostrare il risultato



disegno di poligono

- definire la funzione ***draw_polygon***
  - parametri: numero dei lati, centro e raggio del cerchio circoscritto
- suggerimento:
  - trovare i vertici attorno al centro con ***move_around***
  - unire i vertici per disegnare il poligono





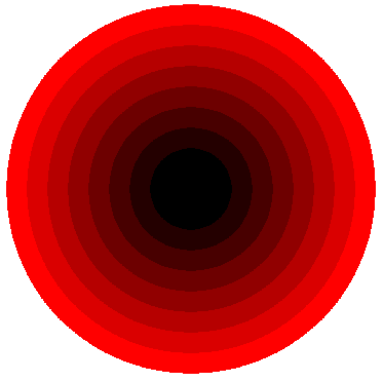
orologio

- definire la funzione ***draw_clock***
  - disegnare 12 tacche a raggiera come in un orologio classico
- miglioramento:
  - disegna anche le tacche dei minuti (più piccole)

UNIVERSITÀ DI PARMA

cerchi concentrici

- chiedere all'utente il ***numero*** di cerchi da disegnare
- ***disegnare*** i cerchi con ***raggio decrescente***, ma tutti con lo ***stesso centro***
- far variare il ***colore*** dei cerchi
  - dal ***rosso*** del livello più ***esterno***
  - fino al ***nero*** del livello più ***interno***



cominciare a disegnare un grosso cerchio rosso
poi, inserire l'operazione di disegno in un ciclo,
togliendo ad ogni passo 10 (p.es.) al raggio e al livello di rosso
infine, determinare automaticamente, prima del ciclo, le variazioni migliori per raggio e colore

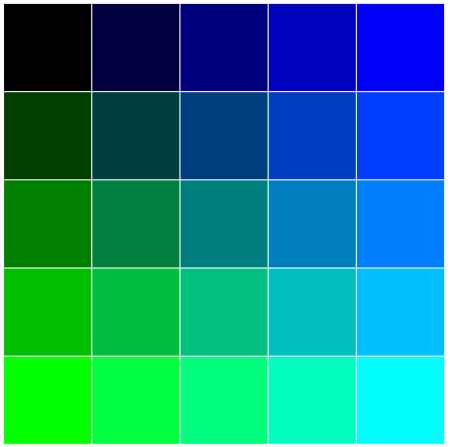
griglia di colori

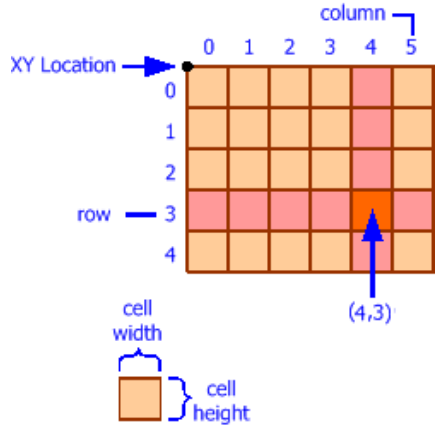
- chiedere all'utente dei valori per ***rows*** e ***cols***
- mostrare una griglia di ***rettangoli*** di dimensione ***rows×cols***
- partire da un rettangolo nero in ***alto a sinistra***
- in ***orizzontale*** aumentare gradatamente la componente di ***blu***
- in ***verticale*** aumentare gradatamente la componente di ***verde***

cominciare a creare una griglia di riquadri tutti neri

con due cicli for annidati

lasciare tra i riquadri un piccolo margine





UNIVERSITÀ
DI PARMA

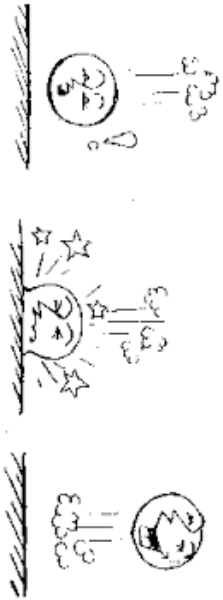
animazioni

esercizi

movimento orizzontale

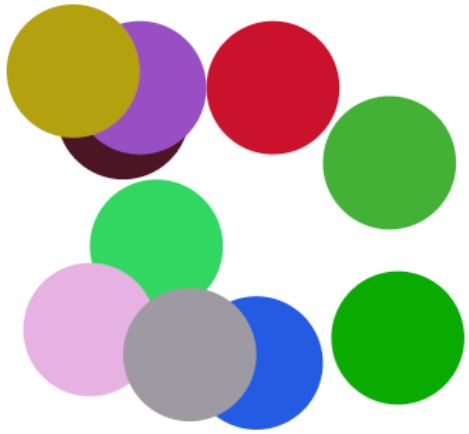
- mostrare una ***pallina*** che si muove in ***orizzontale***
- la pallina ***rimbalza*** sui bordi

memorizzare in una variabile ***dx*** *lo spostamento orizzontale*
da effettuare ad ogni ciclo
cambiare segno a dx quando ***x < 0*** *oppure* ***x + w > screen_width***



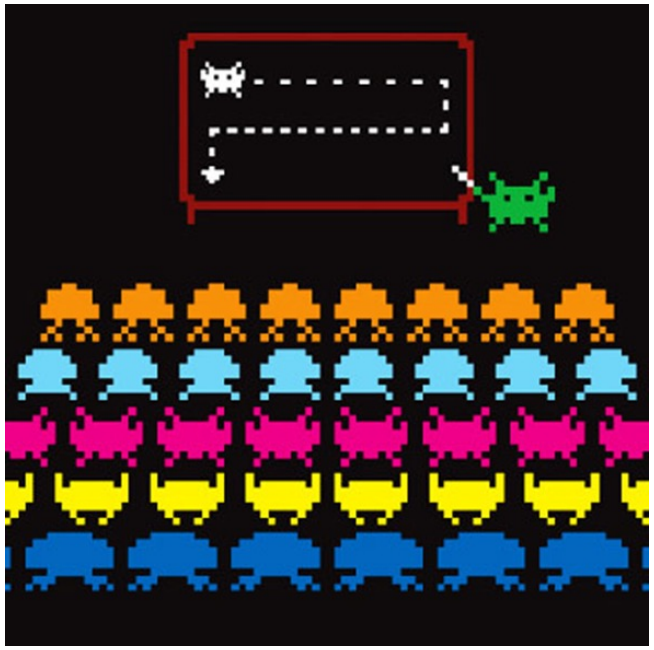
cerchi al click

- definire la funzione ***tick***
  - quando il mouse viene cliccato…
    - se il mouse è «vicino» al centro del canvas…
    - chiedere conferma all'utente e se confermato chiudere l'applicazione
  - disegnare un cerchio nella posizione del click con raggio fisso e colore casuale



movimento a serpentina

- mostrare una ***pallina*** che si muove a ***serpentina***
- partire dall'esercizio precedente
- al momento del **rimbalzo**, imporre un spostamento **verticale**
- fare in modo che, in ogni frame, lo spostamento sia *solo orizzontale*, o *solo verticale*, ma *non diagonale*



poligono rotante

- chiedere all'utente un numero ***n***
- mostrare un poligono regolare di ***n*** lati in rotazione al centro del canvas
- *suggerimento:*
  - riutilizzare la funzione ***draw_polygon*** (esercizio precedente)
  - ruotare variando ultimo l'angolo del primo vertice