NON ENTRI NESSUNO CHE NON CONOSCA LA GEOMETRIA

Motto all'entrata dell'Accademia di Platone

PyAlGeo (PYthon ALGorithmic GEOmetry) è un ambiente di *geometria dinamica* del piano realizzato in Python. Permette di gestire solo poche tipologie di entità geometriche: punti, segmenti, semirette, rette, circonferenze. Tutte le entità sono dinamiche, cioè modificabili interattivamente. Tutte le espressioni (geometriche, numeriche e testuali) sono dinamiche nel senso che vengono ricalcolate (e visualizzate) contestualmente con le modifiche alle entità geometriche coinvolte. È possibile realizzare algoritmi in Python.

Il disegno viene realizzato mediante la grafica della tartaruga oppure mediante la costruzione e manipolazione di entità grafiche (punti, linee, ...).

PyAlGeo permette di sperimentare con la geometria in diversi contesti:

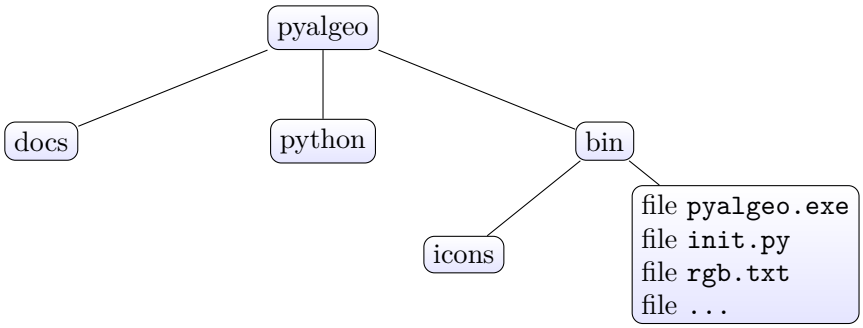
- geometria della tartaruga
- uso di una penna in un sistema cartesiano
- uso di oggetti dinamici (*Point*, *Segment*, *Ray*, *Line*, *Circle*)

Attenzione: questa è una versione provvisoria ed incompleta del manuale; contiene molti errori e ci sono delle funzionalità descritte che non sono state ancora realizzate.

0.1 Installazione

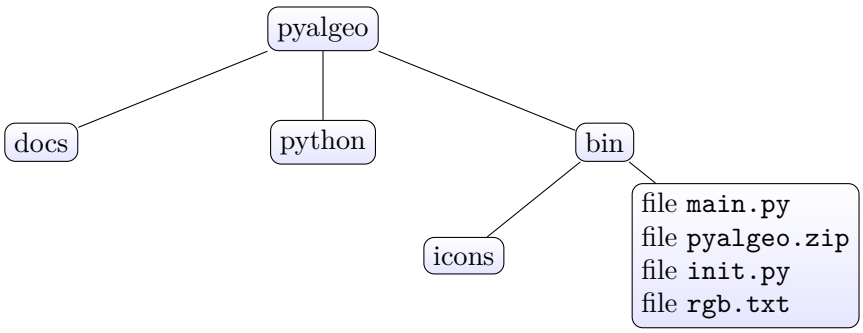
PyAlGeo può essere installato con due modalità diverse:

1. *installazione autonoma*: è sufficiente scaricare il file `ag_inst.zip` è decomprimerlo (in una generica posizione); viene creata la seguente struttura di cartelle:



   Ottenuta questa struttura di cartelle, è sufficiente eseguire il file `pyalgeo` presente nella cartella `bin`; i programmi utente vanno scritti nella cartella `python`.

2. *installazione in modalità di sviluppo*: per questa modalità è necessaria l'installazione dell'ambiente di sviluppo di Python; sono inoltre necessari i seguenti file: `main.py`, `pyalgeo.zip`, `init.py`, `rgb.txt`, `icons.zip`. Il file `icons.zip` va decompresso in una cartella `icons`, mentre il file `pyalgeo.zip` deve essere mantenuto compresso ed importato nel file `main.py`. Una possibile struttura corretta delle cartelle di lavoro è la seguente:

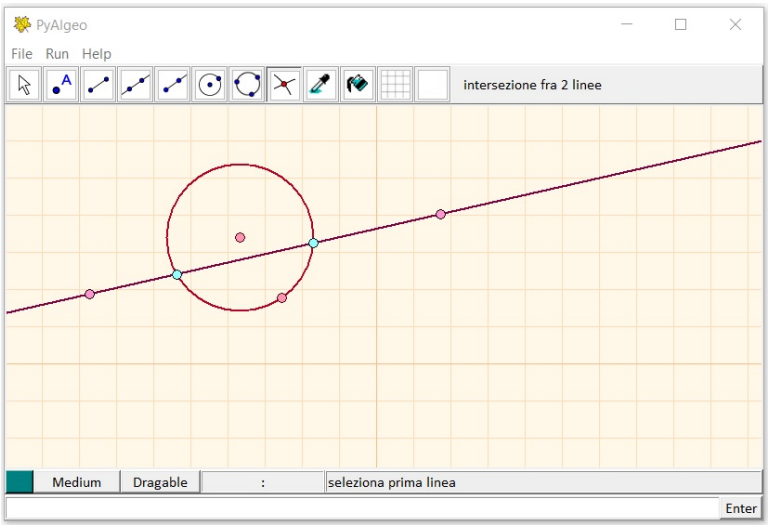


0.2 Modalità d'uso di PyAlGeo

PyAlGeo è un ambiente interattivo e di programmazione per la manipolazione di entità geometriche. Può essere utilizzato in duplice modalità, come descritto a seguire.

0.2.1 PyAlGeo in modalità autonoma

PyAlGeo si attiva mediante lìesecuzione del programma `pyalgeo.exe`. Al momento dell'esecuzione viene eseguito il file `init.py` presente nella cartella del programma; tale file contiene alcune istruzioni (in linguaggio Python) di inizializzazione dell'ambiente. Si presenta la seguente finestra.



Nell'ambiente di questa finestra sono predisposte le seguenti funzionalità:

- attivare le voci di menù corrispondenti alle seguenti funzionalità:
  - *File* > *New* : elimina il disegno presente nella finestra
  - *File* > *Exit* : termina il programma
  - *Run* > *Import* : importa il modulo Python selezionato
  - *Run* > *Exec* : esegue il programma Python selezionato
  - *Help* > *About* : visualizza la versione del programma
  - *Help* > *Manual* : visualizza il manuale del del programma
- creare, modificare, muovere le entità grafiche in modo interattivo, selezionando la specifica icona
- eseguire singole istruzioni sul campo di comando in fondo alla finestra

0.2.2 PyAlgeo in modalità di sviluppo

Questa modalità richiede i seguenti componenti:

- il file `pyalgeo.py` contenente un *main* di esempio di un programma
- il file `pyalgeo.zip` contenente i moduli grafici (compilati) necessari; tale file deve essere importato nel main descritto al punto precedente
- un'installazione dell'ambiente di sviluppo di Python (una versione > 3.5)

Il programma utente si sviluppa attraverso i seguenti passi:

1. scrittura di un programma, in linguaggio Python, contenente le istruzioni da eseguire; tale programma deve includere l'import del file `pyalgeo.zip`
2. esecuzione del programma scritto; viene attivato l'ambiente interattivo, come descritto nel precedente sotto paragrafo

0.3 La grafica della tartaruga

La grafica della t. è integrata all'interno del sistema DyGeo. In *DyGeo* si può utilizzare la geometria della tartaruga con i tradizionali comandi sotto elencati:

- `penup()`: alza la penna dal foglio
- `pendown()`: abbassa la penna sul foglio
- `color(c)`: imposta il colore della penna
- `left(a)`: ruota a sinistra di a gradi
- `right(a)`: ruota a destra di a gradi
- `forward(n)`: avanza di n passi
- `jump(n)`: salta in avanti di n passi
- `hide()`: nasconde la tartaruga
- `show()`: rende visibile la tartaruga
- `home()`: ripristina la condizione iniziale della tartaruga

Gli angoli vengono misurati in gradi. I passi della tartaruga sono misurati in unità virtuali, coerentemente con l'unità di misura del grid che si vede nella finestra grafica.

La t. è un automa con stato, caratterizzato dai seguenti attributi:

- posizione
- angolo di avanzamento
- stato su/giù della penna
- colore della penna

È possibile salvare e ripristinare lo stato mediante i seguenti comandi:

- `push()`: salva in una pila lo stato corrente
- `pop()`: ripristina l'ultimo stato salvato
- `init()`: ripristina lo stato iniziale della t.

Nota. Inizialmente la penna è abbassata sul foglio; pertanto si può evitare il ricorso a *penup* e *pendown*, usando solamente *forward* e *jump*.

0.3.1 La tartaruga nel piano punteggiato

Un modo per dare evidenza grafica alle entità geometriche dei punti consiste nell'unire i punti mediante un segmento; in questo modo i punti diventano lo scheletro portante della figura. Il sistema grafico costituito dalla tartaruga e dalla associata penna può essere manipolato nel piano punteggiato dando vita alla geometria del piano punteggiato; i comandi, che possono essere pensati come rivolti alla penna, sono i seguenti:

- `pos()`: punto corrispondente alla posizione attuale della penna
- `move(P)`: muove la penna sul punto P senza tracciare linea; non modifica la direzione di avanzamento
- `draw(P)`: trascina la penna sul punto P tracciando una linea; non modifica la direzione di avanzamento
- `close()`: chiude la spezzata trascinando la penna sul punto dell'ultima *move*
- `look(P)`: orienta la tartaruga verso il punto P l'orientamento verso il punto
- `dist(P)`: distanza della penna dal punto P (è un *DataObject* dinamico che dipende da P)
- `point()`: disegna un punto alla posizione attuale della penna

Una volta eseguito uno dei comandi descritti sopra, se si sposta il punto P la tartaruga non viene modificata; solo la funzione *dist* ritorna un valore dinamico che dipende dall'eventuale trascinamento del punto P.

Nota. E' possibile ambientare e gestire la tartaruga in un sistema di riferimanto cartesiano (geometria cartesiana) usando le funzioni di *Point* (costruttore e funzioni che ritornano le coordinate di un punto).

Nota. Il disegno della penna e della tartaruga non è né spostabile né agganciabile; per interfacciare la tartaruga con gli elementi grafici (*Line*, ...) bisogna usare le primitive grafiche della tartaruga (quelle elencate sopra) con argomento di tipo *Point* (quelle elencate sopra); in ogni caso il disegno tracciato rimane fisso e non trascinabile.

Esempio 0.3.1 - Un'assegnazione della forma

$$A = pos()$$

ha l'effetto di creare un punto A avente le coordinate uguali a quelle attuali della tartaruga; un eventuale spostamento della tartaruga non modifica il punto A; l'istruzione equivale all'azione di piantare una bandierina con nome A sul punto dove si trova la tartaruga.

0.3.2 La grafica della tartaruga in modalità ad oggetti

La grafica della tartaruga può essere impostata anche secondo la modalità orientata agli oggetti; in particolare:

- per creare una tartaruga t con specifici argomenti di inizializzazione viene richiamato il costruttore della classe ed utilizzata un'assegnazione della forma
$$t \leftarrow Turtle(\dots)$$
Il costruttore *Turtle* può avere degli argomenti quali il colore e lo spessore della penna
- per fare eseguire un'*azione* (comando, settaggio, funzione) viene utilizzata la notazione puntata
$$t.azione(\dots)$$

Esempio 0.3.2 - La seguente porzione di codice disegna di un quadrato rosso di lato di 5 unità:

```
t = Turtle('red')    # oppure t = Turtle()
t.pendown()
t.show()
for _ in range(4):
    t.forward(5)
    t.left(90)
```

Adottando l'impostazione orientata agli oggetti risulta semplice disegnare delle figure composte da parti, ciascuna delle quali viene disegnata da una diversa tartaruga. Se, in qualche modo [1], si muovono contemporaneamente più tartarughe, si ottiene l'effetto visivo del movimento contemporaneo delle varie tartarughe.

Esempio 0.3.3 - In questo esempio viene creato un array di tartarughe che vengono inizialmente orientate verso diverse direzioni, a raggiera; successivamente vengono fatte avanzare contemporaneamente ottenendo l'effetto di una spirale a più rami che si sviluppa dal centro verso l'esterno; l'apparente simultaneità del movimento viene ottenuta avanzando sequenzialmente di poco ciascuna tartaruga

```
n = 17 #numero di tartarughe
t = n*[None]
colors = ['red','green','blue','orange','brown']
for i in range(n):
    t[i] = Turtle()
    t[i].color(colors[i % len(colors)])
    t[i].left(360/n*i)
    t[i].show()
```

[1] Con un ciclo in cui si fanno eseguire brevi spostamenti a tutte le tartarughe oppure associando un *thread* di esecuzione a ciascuna tartaruga.

```
for _ in range(40):
    for i in range(n):
        t[i].forward(.2)
        t[i].left(5)
```

E' possibile far interagire tartarughe diverse.

Esempio 0.3.4 - Nella porzione di programma che segue vengono create due tartarughe: la prima si muove lungo una circonferenza; la seconda si muove orientando ad ogni passo la propria direzione verso la prima, fino a raggiungerla.

```
t1 = Turtle(color='red')
t2 = Turtle(color='blue')
t1.move(Point(-2,1))
t2.move(Point(7,3))
t1.show()
t2.show()
while t1.dist(t2.pos())>.02:
    t1.forward(.1)     # piu' lenta
    t2.forward(.11)    # piu' veloce
    t1.left(3)         # ruota di poco
    t2.look(t1.pos())  # orienta verso t1
```

0.3.3 Estensione della classe Turtle

La classe *Turtle* può essere estesa, aggiungendo ulteriori attributi e nuovi metodi e ridefinendone alcuni di quelli già realizzati. Questa tecnica è tipica della *programmazione ad oggetti* e viene detta *estensione* o *ereditarietà*.

```
class Tarta(Turtle):
    # costruttore
    def __init__(self, col='black'):
        Turtle.__init__(self)
        super().color(col)
        self._passi = 0   # numero di passi percorsi

    # metodo ridefinito
    def forward(self, passi:float):
        super().forward(passi)
        self._passi += passi

    # numero passi - valore dinamico
    def npassi(self):
        return _Value(self.dpos(),lambda:self._passi,'npassi')

    # percorso fatto - valore statico
    def percorso(self):
        return self._passi

    # metodo aggiunto
    def poligono(self, n:int, lato:float):
        for _ in range(n):
            self.forward(lato)
            self.left(360/n)

    # metodo disinnescato
    def color(self, col:str):
        pass

#---- main ----
t = Tarta('red')
t.poligono(7,3)
write('percorso:',t.percorso()) # valore statico
write('npassi:',t.npassi())     # valore dinamico
t.color('blue')   # non ha effetto
k = 1
for _ in range(100):
    t.forward(k)
    t.left(90)
    k += .1
```

0.4 Tipologie di oggetti e loro modalità di generazione

È possibile gestire diverse tipologie di primitive grafiche:

- *Point*: punto
- *Segment*: segmento
- *Ray*: semiretta
- *Line*: retta
- *Circle*: circonferenza

0.4.1 Costruttori degli oggetti grafici

Ogni oggetto grafico viene creato mediante un costruttore che viene attivato interattivamente agendo sulle icone della GUI, oppure mediante l'esecuzione di una specifica istruzione Python avente il nome della classe dell'oggetto (*Point*, *Line*, *Segment*, *Ray*, *Circle*) dell'oggetto che si intende creare e degli argomenti che ne definiscono la geometria e gli attributi grafici. Nel caso in cui il primo argomento sia `INPUT` significa che l'oggetto viene specificato (parzialmente o completamente) interattivamente, agendo sulla GUI; gli attributi grafici (*state*, *width*, *color*) che non vengono specificati esplicitamente, vengono assunti da quelli al momento impostati sulla GUI.

Esempio 0.4.1 - [*costruttore puro*] L'istruzione

```
Line(Point(0,0),Point(1,2),color='red',state=VISIBLE,width=THIN)
```

genera la retta passante per i punti di coordinate $(0,0)$ e $(1,2)$. □

Esempio 0.4.2 - [*costruttore misto*] L'istruzione

```
Segment(INPUT,P,color='blue',state=DRAGABLE)
```

acquisisce il secondo punto di un segmento avente un vertice in P; il segmento assume il colore *rosso*, mentre gli attributi *width* e *state* sono quelli impostati; questi possono essere modificati prima di acquisire in input il secondo punto del segmento. □

Esempio 0.4.3 - [*input puro*] L'istruzione

```
Circle(INPUT)
```

acquisisce in input una circonferenza, con gli attributi correnti. □

0.4.2 Operazioni sugli oggetti grafici

Sulle primitive si possono eseguire diverse tipologie di operazioni:

- accesso alle componenti o proprietà delle primitive:
  - coordinate di un punto
  - estremi di un segmento
  - ...
- operazioni che generano altre primitive vincolate agli argomenti; ad esempio:
  - somma di due punti
  - intersezione fra due rette
  - ...
- entità calcolate mediante espressioni o algoritmi:
  - punto medio (fra due punti)
  - asse di un segmento
  - retta perpendicolare passante per un dato punto
  - bisettrice di un angolo
  - ...

0.4.3 Il valore nullo

Il valore nullo θ rappresenta un valore che è il risultato di un'operazione in un caso estremo, impossibile; esempi:

- intersezione di due rette parallele
- retta individiata da due punti coincidenti
- risultato di un'operazione in cui uno degli argomenti è θ

0.5 Struttura degli oggetti grafici

Ogni oggetto è composto da una sovrapposizione di 3 livelli di informazioni:

1. componente *geometrica*: definisce la *forma* dell'oggetto e ne costituisce l'*essere*
2. componente *grafica*: definisce come l'oggetto *appare* su video
3. componente *dinamica*: definisce come l'oggetto *reagisce* alle sollecitazioni dell'utente

Questi livelli sono specificati mediante degli attributi che definiscono la struttura interna, l'apparenza su video ed il comportemento a seguito di operazioni interattive dell'utente. Questi diversi livelli sono caratterizzati da specifici attributi.

0.5.1 Attributi geometrici

Gli attributi geometrici specificano la forma, la posizione e la struttura degli oggetti grafici e sono

- coordinate geometriche dell'oggetto: dipendono dalla particolare tipologia dell'oggetto: (x, y) per un *Point*, $[(x_1, y_1), (x_2, y_2)]$ per un *Segment* e similmente per le altre categorie di entità

0.5.2 Attributi grafici

Gli attributi grafici definiscono le caratteristiche di visualizzazione. Vengono impostati al momento della creazione dell'oggetto; se non vengono specificati vengono presi quelli di default impostati; possono essere cambiati successivamente alla creazione.

- `color`: colore dell'oggetto; viene specificato mediante una stringa del colore: `'red'`,`'orange'`, `'#FF00A7'`
- `name`: nome dell'oggetto; è un attributo opzionale che viene visualizzato vicino all'oggetto a cui si riferisce, con lo stesso colore dell'oggetto; si sposta contestualmente al movimento dell'oggetto corrispondente
- `width`: spessore della linea: è un numero intero positivo: 1: sottile, 2: media, 3: spessa; al posto dei valori numerici si possono usare le costanti simboliche `THIN`, `MEDIUM`, `THICK`

0.5.3 Attributi di interazione

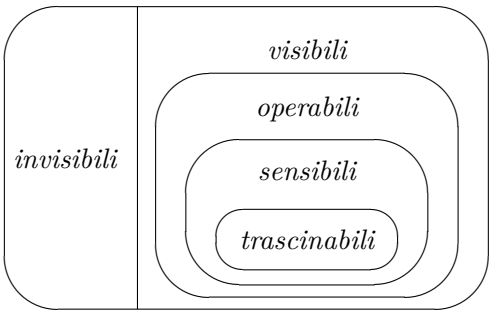
Gli oggetti possono essere manipolati interattivamente; le possibilità dipendono dal loro stato di attivazione specificato dal seguente attributo:

- `state`: stato di attivazione

In ogni istante un oggetto grafico si trova in uno dei seguenti stati di attivazione:

- *invisibile* (`INVISIBLE`=0): oggetto invisibile; quindi non può essere manipolato interattivamente (trascinato, interrogato, modificato nei suoi attributi, ...). Gli oggetti invisibili non sono operabili visivamente (trascinabili, ...) ma possono essere usati mediante programma, per eseguire calcoli
- *visibile* (`VISIBLE`=1): oggetto visibile su video; non può essere oggetto di operazione, agganciato, trascinato o cambiato di formato grafico; esempio: le rette del grid del piano
- *operabile* (`OPERABLE`=2): oggetto visibile su video che può essere argomento di operazione (cancellazione, intersezione, cambio di atributo grafico, ...); non può essere agganciato o trascinato
- *sensibile* (`SENSIBLE`=3): reagisce, cambiando colore, quando viene sovrapposto dal mouse; con un clic può essere selezionato oppure "agganciato" se è attiva la modalità di inserimento; non può essere trascinato
- *trascinabile* (`DRAGABLE`=4): oggetti mobili; possono essere oggetto di operazione, agganciati e trascinati agganciando un loro punto

Si evidenzia il seguente schema insiemistico:



Da questo schema si deducono le seguenti implicazioni:

$$trascinabile \Longrightarrow sensibile \Longrightarrow operabile \Longrightarrow visibile$$

Date queste implicazioni, lo stato di un oggetto può essere specificato mediante un singolo attributo `state` univalore, e sullo stato si possono fare comparazioni di ordine.

Per quanto riguarda la mobilità dei singoli punti si possono avere le seguenti situazioni:

- **punto libero**: può essere mosso liberamente nel piano; viene creato con attibuto di stato `DRAGABLE`
- **punto fisso**: una volta creato non può essere spostato; viene creato con attibuto di stato ≤`DRAGABLE`
- **punto vincolato**: può essere spostato su una linea vincolo
- **punto dipendente**: è in funzione di altri punti; viene generato mediante un'espressione; può essere spostato solo indirettamente muovendo i punti da cui dipende; ad esempio:

```
A = Point(2,3)
B = Point(4,1)
M = (A+B)/2      # punto medio dipendente da A e da B
```

Un oggetto composto, individuato da più punti, può essere modificato e spostato in varie modalità:

- *fisso*: nessuna sua componente può essere spostata
- *modificabile*: se ne possono spostare i singoli punti, cambiandone la forma
- *traslabile*: può essere mosso in modo rigido, senza ruotare
- *ruotabile* se può essere fatto ruotare attorno ad un punto di perno

Componendo i movimenti di base si possono generare altri movimenti:

- ribaltamento (simmetria centrale o assiale)
- ...

Combinando gli attributi di stato delle varie componenti di un'entità grafica si riescono ad ottenere diverse situazioni:

- circolo con centro fisso e raggio variabile: basta creare il centro sensibile (ma non trascinabile) e poi agganciarsi al centro

0.5.4 Modalità di selezione e settaggio degli attributi

In ogni istante sono specificati dei valori correnti degli attributi, visibili sulla GUI; questi attributi. Gli attributi degli oggetti possono essere selezionati e settati con diverse modalità:

1. Il valore degli attributi può essere specificato nel costruttore in fase di creazione dell'oggetto; per default gli oggetti che vengono creati mediante il costruttore hanno l'attributo di stato `INVISIBLE` ed hanno solo le caratteristiche geometriche che servono per i calcoli; per le primitive grafiche che vengono costruite in modo interattivo (argomento `INPUT` del costruttore) i valori degli attributi non specificati nel costruttore vengono assunti da quelli correnti impostati nella finestra grafica dell'applicazione;
2. È possibile acquisire i valori attuali correntemente impostati sulla GUI mediante le funzioni *getcolor*(), *getwidth*(), *getstate*() che ritornano il valore dell'attributo e settare sulla GUI dei valori mediante i corrispondenti comandi *setcolor*(c), *setwidth*(w), *setstate*(s). Questi funzioni e comandi possono essere invocati su singole istanze di oggetti grafici mediante la tradizionale notazione puntata; ad esempio *g.getcolor*() ritorna il colore dell'oggetto grafico g.
3. E' possibile modificare uno o più attributi di un oggetto mediante il metodo d'istanza *config*; ad esempio *g.config*(*color*='red',*state*=`DRAGABLE`) setta il colore rosso ed il valore di stato `DRAGABLE` sull'oggetto g. Il metodo config ha il parametro *mode* che può avere uno dei seguenti due valori:
   - `SINGLE`: (valore di default) viene configurata solo la componente lineare e non eventuali punti base (estremi del segmento, ...)
   - `BROAD`: vengono configurati anche i punti base; in automatico, se si setta l'attributo *state*=`INVISIBLE`, viene attivata la modalità *mode*=`BROAD`
4. E' possibile acquisire e settare i valori degli attributi degli oggetti grafici mediante le icone *getformat* e *setformat*, agendo in modalità interattiva sulla GUI.

Esempio 0.5.1 - La seguente porzione di codice illustra alcune possibilità sulla gestione degli attributi grafici:

```
A = Point((2,3),name='A',state=DRAGABLE)
B = Point((3,1),name='B',color='#EEBB00',state=DRAGABLE)
A.config(name='A1',color='red',state=VISIBLE)
C = Point(INPUT,name='C',msg='punto C?',state=DRAGABLE)
D = Point(INPUT,name='D',msg='punto D?',color='red',state=DRAGABLE)
C.config(color=B.color(),state=A.state(),name=C.name()+'1')
```

0.6 Modi operativi

Durante l'interazione dell'utente mediante mouse in ogni istante è attivo uno dei seguenti modi operativi, ciascuno caratterizzato da una specifica forma del cursore:

- modo *generale*: appare un cursore a forma di freccia; si possono avere diversi effetti a seconda di come si agisce:
  - *spostare elemento*: avvicinandosi ad un elemento grafico trascinabile esso si evidenzia ed è possibile trascinarlo; se l'elemento è vincolato esso si muove sul vincolo; se l'elemento appartiene ad un gruppo, viene spostato l'intero gruppo di elementi
  - *spostare foglio*: puntanto il mouse su un punto vuoto con il tasto sinistro e trascinando si ottiene l'effetto di traslare il foglio

    zoomare: è possibile zoomare ruotando in avanti o indietro la rotellina del mouse; lo zoom avviene attorno al punto corrispondente al cursore del mouse
  - *enquiry*: con il tasto destro si apre un menu contestuale per interrogare o modificare un elemento
- modo *disegno*: appare un cursore a croce; si può inserire una nuova primitiva grafica

In ogni momento è possibile impostare un comando sulla linea di comando.

0.7 Modalità di input degli oggetti

L'input avviene selezionando singoli punti con clic/drag del mouse. In ogni momento è attivo uno dei seguenti modi operativi per l'inserimento delle primitive:

- modo *libero* (*FREE*): i punti vengono presi sul punto dove si fa clic/drag col mouse; l'input di ciascun punto è libero e non si risente della presenza degli altri elementi presenti
- modo *aggancio* (*JOIN*): avvicinandosi il cursore del mouse ad un elemento grafico sensibile, esso si evidenzia e:
  - se si è vicini ad un punto: il punto viene condiviso, diventando un riferimento condiviso ad un punto già presente;
  - se si è vicini ad una linea: il punto selezionato diventa un punto vincolato sulla linea

E' possibile forzare l'input mediante i costruttori delle primitive (vedi `INPUT` ...).

Esempio 0.7.1 - Con l'assegnazione

```
P = Point(INPUT,color='red',state=DRAGABLE)
```

vengono settati sulla GUI i due valori 'red' e DRAGABLE degli attributi di colore e stato, mentre con

```
P = Point(INPUT).config(color='red',state=DRAGABLE)
```

viene eseguito l'input con i valori correnti degli attributi e poi avviene il settaggio dei nuovi valori sul singolo oggetto acquisito, senza modificare sulla GUI i valori correnti degli attributi.

0.8 Modalità di output degli oggetti

Gli oggetti grafici creati, se hanno l'attributo di stato `VISIBLE` (o superiore), vengono visualizzati direttamente sulla finestra grafica. Sulla finestra testo, posta sulla parte sinistra della finestra grafica, è possibile visualizzare, in formato testo, delle stringhe, dei valori e degli oggetti grafici; questi ultimi vengono convertiti in formato testo. Mediante il comando

```
write(s,obj)
```

viene visualizzata la stringa *s* seguita dalla stringa della rappresentazione in formato testo dell'oggetto grafico *obj* (la stringa *s* è opzionale). Ad esempio: `write('Punto P=',P)`. Similmente a *write* si può usare il comando

```
show(s,obj)
```

che visualizza la stringa *s* seguita dalla conversione in stringa dell'oggetto *obj*, con la differenza, rispetto a *write*, che eventuali modifiche all'oggetto vengono direttamente riportate sulla corrispondente stringa sulla finestra testo.

0.9 Assegnazioni, alias e costruttori per copia

Se x è un (riferimento a) un oggetto ($Point$, ...), con l'assegnazione

```
y = x
```

y diventa un riferimento (alias) all'oggetto referenziato da x. Una modifica a y influenza l'oggetto (referenziato da) x.

La funzione $clone(x)$ crea una copia di un generico oggetto x; l'oggetto creato viene solitamente referenziato mediante un identificatore utilizzando la tradizionale istruzione di assegnazione

```
y = clone(x)
```

y diventa così un riferimento ad un oggetto uguale a x ma distinto da esso; un'eventuale modifica a y non interferisce con l'oggetto x.

Un'alternativa all'uso della funzione *clone* consiste nell'uso del *costruttore per copia*: se x è un oggetto di classe C ($Point$,...), $C(x)$ è il *costruttore per copia* che crea un oggetto uguale a x ma distinto da x e con l'assegnazione

```
y = C(x)
```

l'oggetto creato viene referenziato da y.

Esempio 0.9.1 - ...

```
P = Point((2,3),color='black',state=VISIBLE)
s = Segment(P,Point(INPUT,color='red',state=DRAGABLE),name=...)
t = s
t.config(color='blue',name='t',state=DRAGABLE)
t.config(name='newt')
w = Segment(t,color='black',state=DRAGABLE)     #costruttore per copia
w.config(name='W',color='brown',state=DRAGABLE) #espandere il config ai figli
```

Esempio 0.9.2 - ...

```
#---- test con le assegnazioni e clone dei punti ---- [dal file test_point.py]
A = Point(INPUT,name='A',msg='input punto...',color='red',state=DRAGABLE)
B = Point(INPUT,name='B',msg='input punto...',color='green',state=DRAGABLE)
write('dist1(A,B)=',dist(A,B))
T = A   #T e' un alias di A
T.config(name='T',color='orange')
write('dist2(T,B)=',dist(T,B))

#B = A        #B e' un alias di A
#B = Point(A) #B e' un clone autonomo da A
B = Point(T)  #B e' clone autonomo (il vecchio B rimane zombi attivo)

B.config(state=DRAGABLE,name='B',color='blue')
write('dist3(A,B)=',dist(A,B))
```

0.10 Point

I punti sono gli elementi fondamentali dell'ambiente *PyAlGeo*: costituiscono gli elementi grafici, possono essere condivisi fra le primitive e possono essere trascinati e spostati in modalità interattiva.

I punti sono utilizzabili mediante le operazioni elencate a seguire; nella descrizione vengono omessi i parametri opzionali relativi alla specifica degli attributi grafici; se non vengono specificati vengono assunti quelli correnti.

- *costanti*: sono definiti i seguenti identificatori di punti costanti:

  `O`: punto *origine* di coordinate $(0, 0)$
  `U`: punto *unità* di coordinate $(1, 0)$
  `I`: punto *indice* di coordinate $(0, 1)$

- *costruttori* : punti del piano costruibili mediante i seguenti costruttori:

  `Point(P))`: copia del punto P (solo attributi geometrici (e non grafici)
  `clone(P))`: copia integrale del punto P (attributi geometrici e grafici)
  `Point((x,y),name=...,color=...,state=...)`: costruttore di un punto di coordinate (x, y), di specificato *nome*, *colore* e *stato*
  `Point(INPUT,msg=...,on=...)`: acquisizione interattiva con mouse; *msg* (opzionale) specifica il messaggio guida; *on* (opzionale) specifica l'elemento grafico su cui è vincolato il punto
  `Point(RANDOM,on=...)`: generazione casuale di un punto; *on* denota l'eventuale elemento grafico su cui è vincolato il punto; se non viene indicato l'argomento *on* (oppure *on* = *None*) viene selezionato un punto casuale nella finestra correntemente visibile; se viene indicato *on* = (x_1, y_1, x_2, y_2) il punto casuale viene scelto all'interno del rettangolo avente (x_1, y_1) e (x_2, y_2) come vertice in basso a sinistra e vertice in alto a destra
  `Point(PARAM,on=..,param=...)`: punto parametrico sull'elemento grafico *on* con parametro *param* compreso nell'intervallo $[0, 2]$

- *confronti*: un punto può essere confrontato con un altro mediante con i seguenti operatori, coerento con la semantica di Python:

  `A == B`: A e B hanno uguali coordinate geometriche
  `A != B`: A e B hanno coordinate geometriche diverse
  `A is B`: A e B sono riferimenti allo stesso punto
  `not A is B`: A e B sono riferimenti a punti distinti

- *operazioni*: un punto può essere generato mediante un'espressione costituita dalle seguenti operazioni elementari:

  `hom(P,Q,`λ`)`: omotetico di Q rispetto a P di rapporto λ
  `rot(P,Q,`α`)`: ruotato di Q rispetto a P di angolo α
  `med(P,Q)`: punto medio fra P e Q
  `sym(P,Q)`: simmetrico del punto P rispetto a Q
  `sym(P,Q,R)`: simmetrico del punto P rispetto alla retta QR
  `ali(P,Q,R)`: i punti P, Q, R sono allineati
  `par(P,Q,R,S)`: la retta PQ è parallela alla retta RS
  `P + Q`: somma vettoriale fra punti
  `P - Q`: differenza vettoriale fra punti
  `k * P`: moltiplicazione per uno scalare (anche a destra)
  `-P`: opposto del punto (rispetto all'origine)

- *funzioni*: danno come risultato un valore dinamico di classe *Value*:

  `coordx(P)`, `coordy(P)` : coordinate cartesiane del punto P
  `coords(P)` : coppia delle coordinate cartesiane del punto P
  `dist(P,Q)`: distanza fra i due punti P e Q
  `ang(A)`: misura dell'angolo angolo della retta OA
  `ang(A,B)`: misura dell'angolo angolo della retta AB
  `ang(A,V,B)`: misura dell'angolo angolo $A\widehat{V}B$

0.11 Segment, Ray e Line

Segmenti, semirette e rette sono entità strutturalmente ed operativamente equivalenti; si differenziano solo nel modo in cui vengono visualizzate e da come gestiscono i punti vincolati su di esse.

In DyGeo le rette sono rappresentate dagli oggetti della classe *Line*.

- *costruttori* : le rette del piano costruibili mediante i seguenti costruttori:

  `Line(P,Q)`: retta passante per due punti

  `Line(P,ang)`: retta per P e formante un angolo *ang* (in senso antiorario) con l'asse x

  `Line(P,(dx,dy))`: retta per P e per il punto $Q = P+(dx, dy)$; la coppia (dx, dy) definisce l'inclinazione della retta

  `Line('3x-2y+4')` mediante un'equazione carteesiana

  Una retta può essere acquisita interattivamente con il mouse; si possono così avere delle situazioni come le seguenti:

  `Line(INPUT)`: acquisizione interattiva con mouse, selezionando due punti sulla retta

  `Line(INPUT),node=P`: acquisizione interattiva con mouse di una retta del fascio proprio di centro P selezionando due punti sulla retta

  `Line(P,Point(INPUT,on=c))`: acquisizione interattiva con mouse della retta per P e secondo punto selezionato sulla circonferenza c

- *operazioni*:

  `inters(r,s)` : coppia delle coordinate cartesiane del punto di intersezione fra due linee (segmenti, semirette, rette, corconferenze

  `slope(r)` : coefficiente angolare della retta r dato dalla coppia (dx, dy) delle coordinate del punto P della circonferenza unitaria tale che la retta OP sia parallela ad r

- *esempi*:

```
s = Line(T,S).config(color='gray')
C = inters(r,s).config(color='red')
```

Modalità di trascinamento:

- trascinando i punti P e Q che definiscono la retta
- trascinando il primo punto della retta; la retta trasla parallelamente a se stessa

0.12 Circle

…

0.13 Nomi, Text e Label

In PyAlGeo vengono gestite diverse classi di stringhe: *nomi* degli oggetti, *testi* autonomi, *label* degli oggetti.

Nomi

Ogni oggetto grafico che viene creato può avere un nome (attributo `name`); il nome deve essere una stringa; se non viene specificato viene assunta la stringa vuota come nome. Dal nome (se specificato) viene generata automaticamente una label ancorata all'oggetto (nel caso l'oggetto non sia un punto, viene generato automaticamente un punto ancora a cui agganciare la label); la label generata viene mossa contestualmente all'oggetto e mantiene la vicinanza all'oggetto in caso di operazioni di zoom. Un esempio:

```
A = Point(INPUT,name='A'color='green',state=DRAGABLE)
B = Point(INPUT,color='blue',state=DRAGABLE)
C = (A+B).config(state=DRAGABLE,color='red',name='A+B')
#C = (A+B).config(state=DRAGABLE,color='red',name=str(coordx(A)))
```

Testi

Un testo è un oggetto grafico autonomo di tipo stringa posizionato in un punto del piano. Un volta creato, il testo si stacca dal punto ed ha una vita separata (in drag). Un testo può essere dinamico, avendo dei parametri `{0}`, `{1}`, ... che vengono istanziati dai valori specificati nel successivo parametro `variables`.

```
Text((5,6),"Testo non dragabile",color='blue',state=VISIBLE)
Text((5,5),"Testo dragabile rosso",color='green',state=DRAGABLE).config(color='red')
Text((5,4),"TESTO parametrico = {0}",123.45, color='brown').config(state=DRAGABLE)
P = Point(5,3,color='brown',state=DRAGABLE)
t=Text(P,"Px={0}",coordx(P),color='orange').config(state=DRAGABLE)
Text(Point(5,2),'P:[{0},{1}]',[coordx(P),coordy(P)],color='orange',state=DRAGABLE)
```

Label

Una label è un testo che viene agganciato ad un oggetto e rimane solidate con esso. Una label può essere dinamica (similmente ai testi dinamici). In zoom una label rimane alla stessa distanza (in pixel) dall'oggetto al quale è agganciata. Una label è dragabile indipendentemente dall'oggetto a cui è ancorata. Creare label su punto equivale (quasi) a creare punto con nome (a parte i colori). Un esempio:

```
Point(1,1,name='PUNTO(1,1)',color='red',state=DRAGABLE)
Label(Point(2,2,color='red',state=DRAGABLE),"LABEL(2,2)",color='brown',state=DRAGABLE)
Label(B,"B=[{0},{1}]",[coordx(B),coordy(B)],color='brown').config(state=DRAGABLE)
```

0.14 Piani di lavoro sulle primitive

AlgoGeo gestisce le primitive su 3 piani:

- piano geometrico degli oggetti in memoria (file `graph.py`, contenente varie classi), con attributi geometrici e grafici; gestisce le aggregazioni (connessioni) fra gli oggetti. In questo piano vengono svolti i *calcoli*.
- piano grafico (file `plane.py`, contenente le classi `Pen`, `Tarta`): gestisce i sistemi di coordinate e trasformazioni; gestisce il componente grafico (canvas) per la visualizzazione grafica. In questo piano vengono svolte le operazioni di visualizzazione grafica su video.
- piano interattivo di input dell'utente (file `iplane.py`): gestito mediante programmazione in linguaggio Python. In questo piano vengono svolte le interazioni con l'utente, mediante i dispositivi di input (tastiera, mouse, ...).

0.15 Note varie

Nota. E' interessante analizzare da un punto di vista implementativo la relazione esistente fra la penna e la tartaruga; si possono individuare (almeno) due relazioni:

- la *tartaruga usa una penna*; la penna non può essere usata direttamente ma i comandi banno indirizzati alla tartaruga che ha la diretta responsabilità di come gestire la sua penna
- la *tartaruga è una penna* dotata di un meccanismo che permette di mantenere e gestire una direzione di avanzamento; in questo caso il sistema tartaruga+penna consente di essere comandato sia mediante i comandi della tartaruga che quelli della penna

Nota. In DyGeo i punti sono dinamici e possono essere spostati interattivamente mediante il mouse; compatibilmente con questo fatto, ci sono dei comandi per la tartaruga che recepiscono questa dinamicità dei punti:

- `join(P)`: sposta ed aggancia la penna sul punto P; se il punto viene spostato, la penna mantiene l'aggancio con il punto; così consente il disegno a mano libera mediante il mouse
- `look(P)`: orienta la tartaruga verso il punto P; se il punto P viene spostato, la tartaruga continua a mantenere l'orientamento verso il punto

Quanto sopra non è stato implementato in quanto rimane il problema: *Cosa fare quando su muove la t. che è stata agganciata o rivolta verso un punto*?

Nota. Obiettivi di DyGeo:

- deve essere elementare: con poche primitive, facile da usare (non deve competere con GeoGebra o simili)
- deve essere dinamico: oggetti muovibili in modo interattivo, con visualizzazione testuale *dinamica*
- deve essere integrabile all'interno di programmi Python oppure mediante l'esecuzione di script Python nell'apposita finestra di DyGeo

0.16 Primitive grafiche

In DyGeo sono previste le seguenti tipologie di primitive grafiche:

- **Point**: punto identificato da due coordinate
- **Segment**: segmento di due dati estremi
- **Line**: retta passante per due dati punti
- **Circle**: circonferenza dato il centro e un punto o il centro e un raggio
- **Ray**: semiretta di dato vertice e punto di passaggio
- **Text**: testo libero

Gli oggetti geometrici sono composti da una componente *geometrica* e da una componente *grafica*.

0.17 Operazioni sulle figure

Le operazioni sulle primitive possono essere denotate nei seguenti modi:

- funzionale
- con operatore infisso

Nella notazione funzionale si ha la seguente struttura di base:

$$op(arg_1, arg_2, \ldots, arg_n)$$

dove $arg_1, arg_2, \ldots, arg_n$ sono gli argomenti geometrici necessari a generare il risultato. *op* può essere il nome di una classe oppure il nome di un'operazione.

Ogni operazione ritorna un oggetto grafico con gli attributi posti uguale a quelli di default correntemente impostati. È possibile modificarli con una delle seguenti modalità:

- precisando gli attributi nel costruttore:

```
P = Point(2,3,color='red',name='P',)
```

- mediante il metodo `config`:

```
Inters(p1,p2).config(color='red',name='T')
```

- specificando, al momento della craezione, un "pacchetto" di attributi:

```
cfg = config(color='blue',width=5)
p = Point(3,4,cfg)
```

0.18 Valori ed espressioni dinamiche

In un ambiente di geometria dinamica è necessario disporre di *espressioni dinamiche* legate a delle quantità che dipendono da elementi geometrici (distanza, lunghezza, ...) che variano dinamicamente. Ciò deve valere sia per i numeri che per gli oggetti grafici. Di conseguenza gli attributi geometrici sono dinamici.

In PyAlGeo i punti e le altre antità grafiche sono modificabili dinamicamente; di conseguenza vengono modificati anche i valori numerici e logici che dipendono dalle entità grafiche. Un *Value* è un valore che dipende da altre entità grafiche; una modifica alle entità grafiche aggiorna automaticamente i valori che dipendono da esse.

Esempio 0.18.1 - Sono oggetti di classe *Value* i seguenti: le coordinate dei punti, la distanza fra due punti.

Esempio 0.18.2 - Un cerchio avente come raggio la distanza fra due punti, si automodifica se vengono spostati i punti che definiscono la distanza.

Se v è un valore dinamico, con $val(v)$ si ottiene il valore attuale (statico) di v.

Un *DynExp* è un'espressione (denotabile anche mediante un identificaore) il cui valore è legato ad un'espressione i cui argomenti sono funzione degli elementi geometrici. Se vengono spostati, l'espressione modifica contestualmente il suo valore e tutto viene riportato sul video. DynExp può essere anche una primitiva grafica (ad esempio il punto medio fra due dati punti).

Il tipo di un'espressione dinamica può essere:

- un oggetto grafico; ad esempio: 1.4*P
- un valore numerico; ad esempio: dist(P,Q)

Un'espressione coinvolgente almeno un argomento modificabile (ad esempio un punto spostabile) è un'espressione dinamica. Un'espressione dinamica prevede un metodo *eval* che ritorna il valore dell'espressione in funzione del valore attuale degli argomenti coinvolti nell'espressione. Ogni modifica ad un argomento modificabile causa un'immediata rivalutazione di tutte le espressioni dinamiche che dipendono dal dato argomento, e, nel caso queste espressioni dinamiche siano visibili, il valore attuale viene visualizzato.

0.19 Valori e punti condizionali

I *valori condizionali* sono oggetti della forma

$$CondValue(c, x, y)$$

dove c è una espressione logica dinamica ed x ed y sono delle generiche espressioni (anche dinamiche).

Usando i valori condizionali si possono costruire degli *oggetti condizionali*. (cfr. *CondPoint* ...)

0.20 Struttura organizzativa degli oggetti grafici

Gli oggetti grafici sono strutturati ed organizzati mediante dei legami (*parents*, *children*) ...

I *punti* costituiscono gli atomi degli oggetti grafici ...

La composizione può essre fatta in vari modi:

- in fase di costruzione, agganciando un punto ad un altro
- dopo aver costruito le singole componenti, vengono agganciate fra di loro in raggruppamenti (meccanismo simile a raggruppa/separa di *Word*)

0.21 Sistemi di riferimento

- **coordinate evento** (event.x, event.y); descrivono l'evento clic del mouse; (0,0) è sempre nell'angolo alto a sinistra della finestra di disegno, anche se il canvas viene scrollato (coordinate pixel); sono coordinate che descrivono l'interazione con l'utente [*coordinate naturali*]
- **coordinate grafiche**: usate per disegnare sul canvas (usate dalle funzioni di tkinter: `canvas.create_oval(...)`; dopo uno scroll in basso ed a destra, l'origine $(0, 0)$ viene spostata all'interno del canvas [*coordinate intere*]
- **coordinate cartesiane**: coordinate virtuali utente (window); usate per descrivere la geometria degli oggetti; [*coordinate reali*]

0.22 Attributi di implementazione degli oggetti grafici

0.22.1 Attributi di struttura

Questi attributi definiscono la struttura organizzativa dei dati:

- `id`: numero identificativo di ciascun oggetto elementare; corrisponde alla primitiva grafica di *tkinter*
- `parents`: lista degli *id* degli oggetti padre (muovendo i quali si muove automaticamente gli oggetti figli)
- `children`: lista degli *id* degli oggetti figli (muovendo l'elemento si muovono automaticamente i figli)

0.22.2 Attributi di classificazione

- `type`: tipo dell'oggetto (classe di graph.py: *Point*, *Circle*, ...)
- `name`: stringa che individua univocamente l'oggetto grafico; può servire per la costruzione di espressioni, ...
- `label`: etichetta testuale associata all'oggetto
- `text`: testo che compare nella label (*ObjLabel*) dell'oggetto (può dipendere da valori variabili (*DataObject*)); viene solitamente inizializzato uguale a *name*
- `tags`: etichette associate all'oggetto (livello, . . .)

1

NOTE: DYNAMIC GEOMETRY

1.1 Note varie

...

1. L'uso di un software di geometria dinamica è importante in campo didattico in quanto consente di creare delle situazioni che favoriscono la formulazione di congetture che poi possono essere sperimentate, analizzate e concluse con una dimostrazione teorica con l'uso di strumenti e tecniche logiche e formali. [lc-2set21]
2. Il costruttore per copia consente di gestire il *tracing* ed il *freezing*, in alternativa a quanto segue.

   E' possibile evidenziare il movimento dei punti mediante il comando *trace*: per un generico punto `P` i comandi sono i seguenti:

   - `P.trace(colore)` : si attiva la scia di movimento
   - `P.trace()` : disattiva la scia di movimento

3. Altre possibilità per settare il valore delle primitive grafiche:

   - si possono creare pacchetti di attributi ed impostarli alle entità grafiche:

     ```
     c1 = Conf(color='red',state=DRAGABLE)
     c2 = Conf(color='blue',state=SENSIBLE)
     A.setConf(c1)
     ```

   - di una primitiva si può ottenere il pacchetto di configurazione:

     ```
     A = Point(...)
     B = Point(...)
     c = A.getConf()
     B.setConf(c)
     ```

1.2 I moduli del codice

1. `DyGeo.py`: programma da eseguire in Python; il main finale permette di gestire diverse modalità; costruisce il sistema grafico; crea alcuni oggetti globali (*Origine*, *_wg*, *_tur*)

2. `kinter.py`: funzioni di interfaccia
   - data dell'intestaatzione della GUI
   - inizializza i componenti della grafica (*kinit*)
   - crea la finestra grafica (*WinGra*)
   - interfaccia le funzioni della grafica della t.
   - interfaccia tutte le funzioni utente
   - definisce funzioni *Point*, *Segment* (wrapper delle omologhe *KPoint*, ... di graph.py)

3. `wingra.py`: organizza la finestra grafica
   - gestisce la linea di comando
   - definisce la funzione *input* che richiama *point*, *segment*, ...

4. `iplane.py`:
   - gestione degli eventi
   - gestione del rubberband (in *OnCanvasMouseMove*)

5. `plane.py`:
   - richiama tkinter definendo *draw_point* richiamando *tk.create_oval*
   - gestione dei cambi di coordinate
   - implementa il *grid*

6. `panel.py`: analogo del *Panel* di Java e del *Canvas* di tkinter;
   - wrapper verso le librerie grafiche (richiama *tkinter*)
   - contiene le funzioni *draw_line*, *draw_point*, *draw_oval*, ...
   - definisce i colori di quando avviene *mouse-over*
   - gestisce *itemconfig*

- gestisce *zoom* e *drag*

7. `graph.py`: definizione degli oggetti grafici e dati:
   - definisce le classi degli oggetti grafici (*GraphicObject*, *DataObject*, ..., *KLine*, *KSegment*, ...)
   - definisce metodo *config* degli oggetti grafici

8. librerie grafiche Python: tkinter, PyGTK, ...

1.3 Modi operativi - versione vecchia (fino al 25/7/21)

Durante l'interazione dell'utente mediante mouse in ogni istante è attivo uno dei seguenti modi operativi, ciascuno caratterizzato da una specifica forma del cursore:

- modo *comando*: modalità generica in cui si deve impostare un comando sulla linea di comando o impostare un altro modo
- modo *drag*: avvicinandosi ad un elemento grafico trascinabile esso si evidenzia ed è possibile trascinarlo; se l'elemento è vincolato esso si muove sul vincolo; se l'elemento appartiene ad un gruppo, viene spostato l'intero gruppo di elementi
- modo *pan/zoom*: per spostare l'intero piano o per zoomare (è possibile zoomare ruotando la rotellina del mouse)
- modo *select*: selezione cumulativa di elementi visibili; con un clic con il tasto sinistro del mouse si seleziona un elemento, con il tasto destro si apre un menu di scelte sul dato elemento avvicinandosi ad un elemento, esso si evidenzia e con clic lo si seleziona; con il tasto *Ctrl* premuto si incrementa la lista degli elementi selezionati; gli elementi selezionati possono essere sottoposti a successive operazioni, quali:
  - interrogazione sugli attributi dell'elemento
  - aggregazione di più elementi in gruppo
  - modifica degli attributi
  - trascinamento (se tutte le componenti del gruppo sono trascinabili)
  - eliminazione
- modo *insert*: per inserire in input primitiva grafiche

1.4 Note implementative

- DyGeo non deve essere una sorta di editor grafico off-line
- Per controllare se un oggetto x è (almeno) visibile:

```
if x._state ≥ VISIBLE: ...
```

1.5 Sul segno di uguaglianza

Il tradizionale segno di uguaglianza =, a seconda dei contesti d'uso, assume dei significati completamente distinti. Per evitare ambiguità, differenziamo tale segno secondo le casistiche descritte a seguire.

- $e_1 = e_2$: è una condizione che esprime un'uguaglianza fra le due espressioni e_1 ed e_2; tale uguaglianza può essere vera o falsa, oppure vera solo per alcune istanziazioni delle variabili che compaiono nelle due espressioni; esempio: $x^2 - 4x + 4 = 0$
- $e_1 \equiv e_2$: esprime l'identità fra le due espressioni e_1 ed e_2, ossia e_1 ed e_2 sempre uguali fra loro, per qualsiasi istanziazione di valori alle variabili che compaiono nelle due espressioni; esempio: $(x+1)^2 \equiv x^2 + 2x + 1$
- $x \leftarrow e$: assegnazione all'identificatore x del valore ottenuto dalla valutazione dell'espressione e; esempio: $a \leftarrow b * h$
- $f(x) \stackrel{\text{def}}{=} e$: definizione della funzione f, di argomento x, con quanto specificato nell'espressione e; esempio: $area(x, y) \stackrel{\text{def}}{=} x * y$
- $e_1 \rightarrow e_2$: l'espressione e_1, eseguendo dei calcoli, viene trasformata nell'espressione e_2; la scrittura può trovarsi anche in una catena di trasformazioni; esempio: $(7-2) * 4 \rightarrow 5 * 4 \rightarrow 20$