

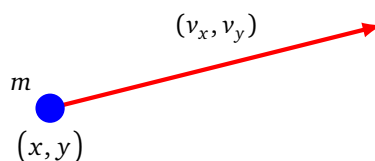
# Mouvement de particules

Tu vas simuler le mouvement d'une particule soumise à différentes forces, comme la gravité ou des frottements. Tu appliqueras ceci afin de simuler le mouvement des planètes autour du Soleil. Cette activité utilise la programmation objet.

## Cours 1 (Une particule).

**Modélisation.** Une particule est modélisée par cinq valeurs :

- ses coordonnées  $x$  et  $y$ ,
- les coordonnées  $v_x$  et  $v_y$  de sa vitesse,
- sa masse  $m$ .



**Particule soumise à aucune force.** Si aucune force n'agit sur la particule, alors elle conserve sa direction et sa vitesse. Ainsi à l'instant élémentaire suivant, la nouvelle position de la particule est :

$$\begin{cases} x' &= x + v_x \\ y' &= y + v_y \end{cases}$$

La particule a donc un mouvement rectiligne uniforme.

*Preuve.* La vitesse est la dérivée de la position, ainsi par exemple  $v_x$  est la limite du taux d'accroissement  $\frac{x(t+dt)-x(t)}{dt}$ . En considérant que  $dt$  est une durée infinitésimale on obtient ;  $x(t + dt) = x(t) + v_x dt$ . En choisissant comme unité de temps  $dt = 1$ , on obtient la formule voulue. Les calculs sont identiques pour  $v_y$ .

**Particule soumise à une force.** Si la particule est soumise à une force  $\vec{F}$  dont les composantes sont  $(F_x, F_y)$  alors à l'instant élémentaire suivant, la nouvelle **vitesse** de la particule est :

$$\begin{cases} v'_x &= v_x + F_x/m \\ v'_y &= v_y + F_y/m \end{cases}$$

Comme la vitesse est modifiée, cela induit un changement sur la future position.

*Preuve.* Le principe fondamental de la mécanique affirme que :

$$\vec{F} = m\vec{a}$$

où  $\vec{a}$  est le vecteur accélération et  $m$  la masse. L'accélération est la dérivée de la vitesse, donc en coordonnées on a :

$$F_x = m \frac{dv_x}{dt} \quad F_y = m \frac{dv_y}{dt}$$

Autrement dit  $m \frac{v_x(t+dt) - v_x(t)}{dt} = F_x$ . Donc  $v_x(t+dt) = v_x(t) + \frac{F_x}{m} dt$ . En normalisant à  $dt = 1$ , on obtient la formule voulue.

S'il y a plusieurs forces  $\vec{F}_1, \vec{F}_2 \dots$  alors on les regroupe en un seul vecteur, la force résultante :  $\vec{F} = \vec{F}_1 + \vec{F}_2 + \dots$

### Activité 1 (Une particule).

*Objectifs : programmer le mouvement d'une particule et son affichage.*

Informatiquement, on code une particule par un objet de la classe `Particule()` contenant 5 attributs  $x, y, v_x, v_y, m$  (correspondant à  $x, y, v_x, v_y, m$ ). Voici le début de la définition de la classe `Particule()` que tu vas peu à peu compléter :

```
class Particule():
    def __init__(self, x, y, vx, vy, m):
        self.x = x
        self.y = y
        self.vx = vx
        self.vy = vy
        self.m = m

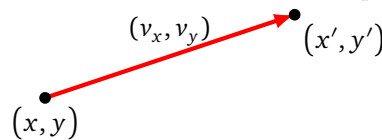
    def __str__(self):
        ligne = "(" + str(self.x) + ", " + str(self.y) + ")",
        (" + str(self.vx) + ", " + str(self.vy) + ")", " + str(self.m)
        return ligne
```

Voici un exemple d'initialisation d'une particule  $p$  placée en  $(-100, 100)$  avec une vitesse de vecteur  $(20, 0)$  (donc horizontale) et de masse  $m = 1$ . La méthode `__str__()` définie plus haut permet d'afficher proprement l'objet  $p$ .

```
p = Particule(-100, 100, 20, 0, 1)
print(p)
```

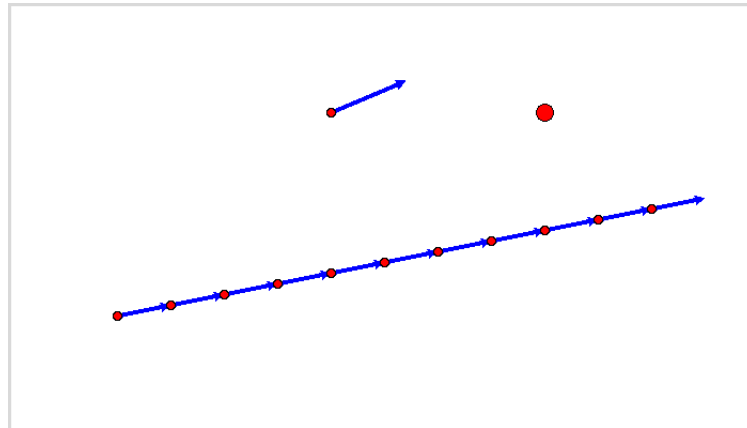
1. Complète la définition de la classe avec une méthode `action_vitesse(self)` qui déplace la particule en suivant le vecteur vitesse. Les nouvelles coordonnées sont données par la formule :

$$\begin{cases} x' &= x + v_x \\ y' &= y + v_y \end{cases}$$



2. Complète la définition de la classe avec une méthode `affiche(self)` qui affiche graphiquement la particule.

Sur la figure ci-dessous en haut à gauche une particule avec son vecteur vitesse, en haut à droite une particule sans affichage de son vecteur vitesse mais de masse plus grosse. En bas une particule qui se déplace suivant son vecteur vitesse (sur 10 unités de temps, en suivant un mouvement rectiligne uniforme).

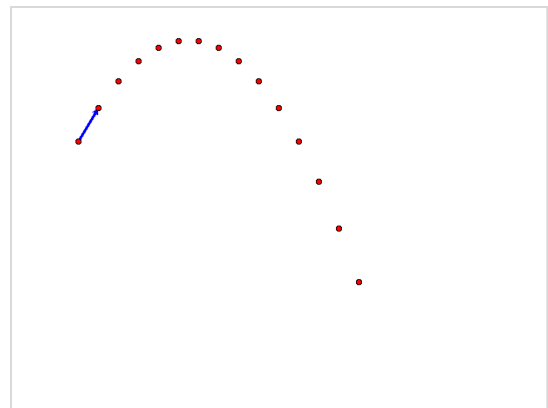
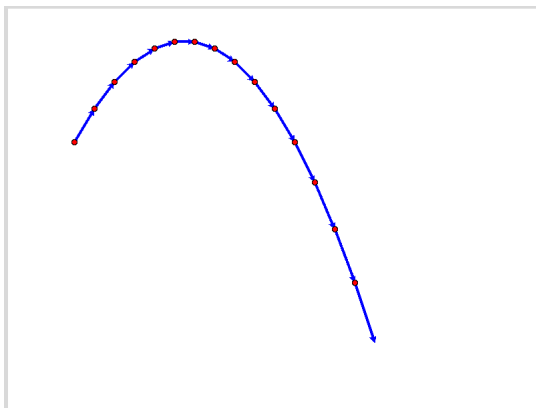


### Indications.

- Utilise le module `tkinter` (voir plus bas) (avec une unité qui vaut un pixel, le point (0,0) étant au centre de l'écran). Le passage des coordonnées réelles aux coordonnées graphiques se fait par les formules  $i = \text{Largeur}/2 + x$  et  $j = \text{Hauteur}/2 - y$ .
  - Tu peux définir ta fonction avec une entête `affiche(self, avec_fleche=False)` et laisser le choix de l'affichage du vecteur vitesse sous la forme d'une flèche.
  - Le rayon du disque peut dépendre de la masse.
3. Complète la définition de la classe avec une méthode `action_gravite(self, gravite=0.2)` qui correspond à l'action de la force de gravité sur la particule. Cela correspond à changer la valeur de la vitesse verticale, la nouvelle valeur  $v'_y$  étant calculée à partir de l'ancienne valeur  $v_y$  par la formule :

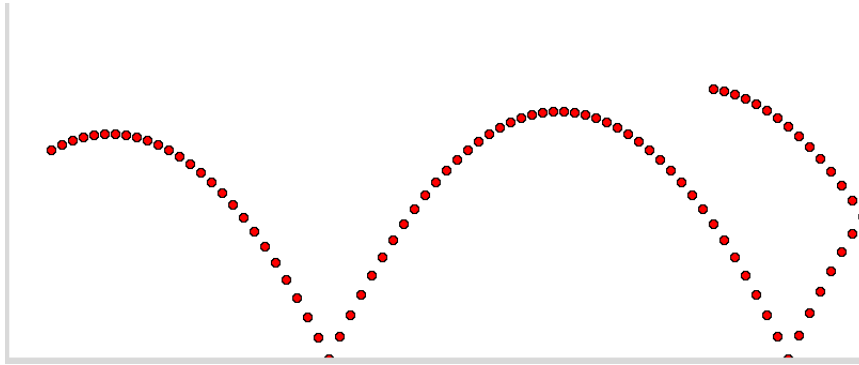
$$v'_y = v_y - g.$$

Teste différentes valeurs de la constante de gravité.



Comme on peut le voir sur les formules (et sur les tracés) il est remarquables que les mouvements de particules de deux masses différentes sont identiques.

4. Complète la définition de la classe avec une méthode `rebondir_si_bord_atteint(self)` qui empêche la particule de sortir de l'écran. Pour cela si  $x$  est trop grand ou trop petit alors inverse le signe de  $v_x$  (c'est-à-dire  $v_x \leftarrow -v_x$ ). De même pour  $y$ .



5. Complète la définition de la classe avec une méthode

`action_frottement(self, frottement=0.005, exposant=2)`

qui correspond à l'action d'une force de frottement.

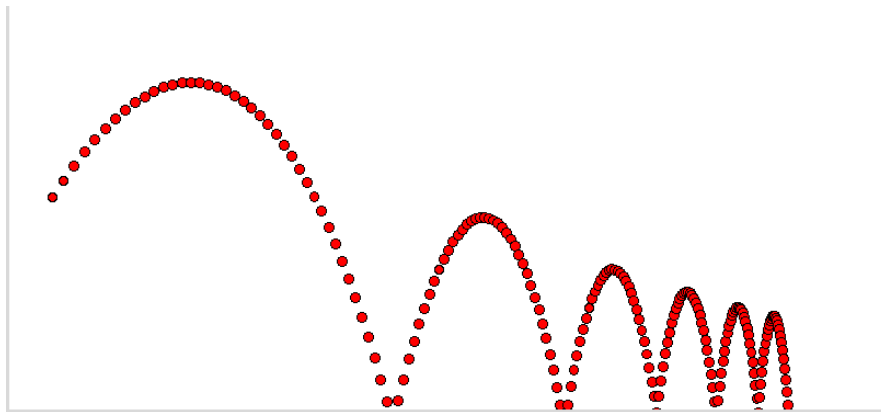
On note

$$v = \sqrt{v_x^2 + v_y^2}.$$

Les nouvelles valeurs du vecteur vitesse sont données par les formules :

$$\begin{cases} v'_x &= v_x - \frac{f}{m} \cdot v^e \cdot \frac{v_x}{v} \\ v'_y &= v_y - \frac{f}{m} \cdot v^e \cdot \frac{v_y}{v} \end{cases}$$

où  $f$  est le coefficient de frottement et  $e$  l'exposant de frottement. Essaie différentes valeurs du coefficient de frottement et surtout de l'exposant. Un exposant  $e = 1$  correspond à un frottement pour une particule se déplaçant à faible vitesse ; un exposant  $e = 2$  correspond à une vitesse élevée ; des exposants  $1 < e < 2$  sont possibles.



6. Complète la définition de la classe avec une méthode `mouvement(self)` qui regroupe la succession des actions définies : `action_vitesse()`, `action_gravite()`, `action_frottement()`, `rebondir_si_bord_atteint()` Ainsi pour simuler le mouvement d'une particule il suffit d'écrire :

```
# Constantes pour l'affichage
```

```
Largeur = 800
```

```
Hauteur = 600
```

```
# Fenêtre tkinter
```

```
from tkinter import *
```

```
root = Tk()
```

```

canvas=Canvas(root,width=Largeur,height=Hauteur,background="white")
canvas.pack(side=LEFT, padx=5, pady=5)

p = Particule(-300,10,5,2,10)
for k in range(100):
    p.mouvement()
    p.affiche()

root.mainloop()

```

### Activité 2 (Particules en mouvement).

*Objectifs : afficher une ou plusieurs particules en mouvement.*

Voici le « film » de 10 particules, chacune étant lancée à partir du même point mais avec un vecteur vitesse initial différent.



Programme tout cela en utilisant les explications qui suivent.

Voici une classe `TkParticule()` héritée de la classe `Particule()` qui permet d'afficher une particule en mouvement (la particule évolue sans afficher sa trace). Il y a deux attributs supplémentaires : un attribut pour la couleur, un attribut pour l'identifiant `tkinter` de la particule qui permet ensuite à la méthode `affiche()` de déplacer le disque grâce à la méthode `move()` de `tkinter`.

```

class TkParticule(Particule):
    def __init__(self,x,y,vx,vy,m,couleur="red"):
        Particule.__init__(self,x,y,vx,vy,m)
        self.couleur = couleur
        i,j = xy_vers_ij(x,y)
        rayon = min(max(1,m),10)
        # Création de l'objet tkinter
        disque = canvas.create_oval(i-rayon,j-rayon,

```

```

        i+rayon,j+rayon,fill=self.couleur)
    self.id = disque

```

```

def affiche(self):
    canvas.move(self.id,self.vx,-self.vy)

```

La fonction `xy_vers_ij(x,y)` transforme les coordonnées réelles  $(x,y)$  en coordonnées graphiques  $(i,j)$ .

Voici un lancé de plusieurs particules (on utilise le module `time`).

```

liste_particules=[TkParticule(-300,0,10,j,5,couleur=hasard_couleur())
                  for j in range(10)]
for k in range(200):
    for p in liste_particules:
        p.mouvement()
        p.affiche()

    canvas.update()
    sleep(0.05)

root.mainloop()

```

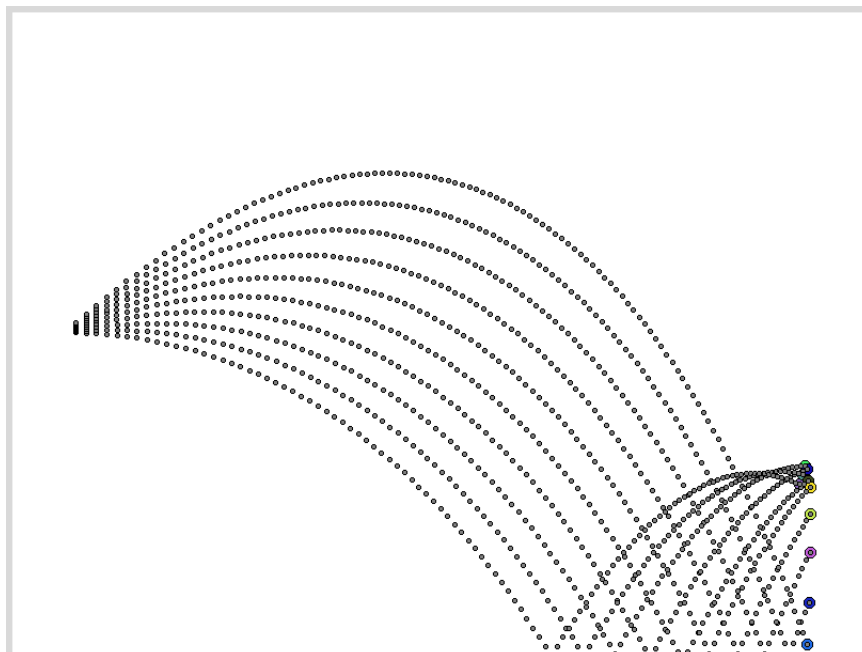
Voici une fonction qui renvoie une couleur au hasard (en utilisant le module `random`).

```

def hasard_couleur():
    R,V,B = randint(0,255),randint(0,255),randint(0,255)
    couleur = '#%02x'%02x'%02x' % (R%256, V%256, B%256)
    return couleur

```

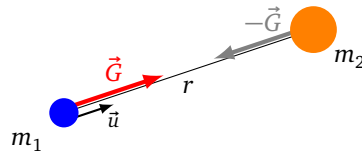
Une autre idée est de garder la trace des positions précédentes.



**Cours 2** (Attraction gravitationnelle).

Deux astres s'attirent selon la force d'attraction gravitationnelle :

$$\vec{G} = G \frac{m_1 m_2}{r^2} \vec{u}$$



où

- les astres sont de masses  $m_1$  et  $m_2$ ,
- $r$  est la distance entre les deux astres,
- $G$  est la constante de gravitation universelle (on prendra arbitrairement  $G = 100$ ),
- et  $\vec{u}$  est un vecteur unité pointant de l'astre 1 vers l'astre 2.

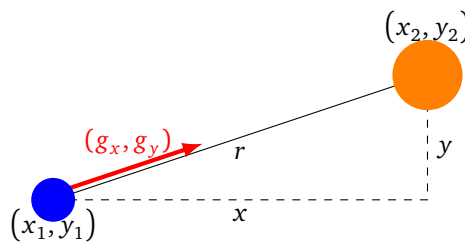
Avec ces notations  $\vec{G}$  est la force d'attraction de l'astre 2 agissant sur l'astre 1, et  $-\vec{G}$  est la force d'attraction de l'astre 1 agissant sur l'astre 2.

On note  $(x_1, y_1)$ ,  $(x_2, y_2)$  les coordonnées des astres. On pose :

$$x = x_2 - x_1 \quad y = y_2 - y_1 \quad r = \sqrt{x^2 + y^2}$$

Les coordonnées du vecteur  $\vec{G}$  sont alors :

$$g_x = G \frac{m_1 m_2}{r^2} \cdot \frac{x}{r} \quad \text{et} \quad g_y = G \frac{m_1 m_2}{r^2} \cdot \frac{y}{r}$$

**Activité 3** (Mouvement des planètes).

Objectifs : simuler le mouvement des planètes autour du Soleil grâce à la force de gravitation.

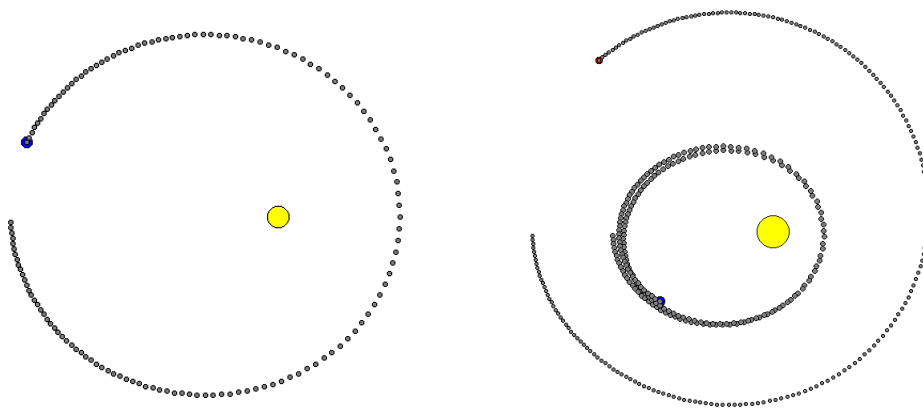
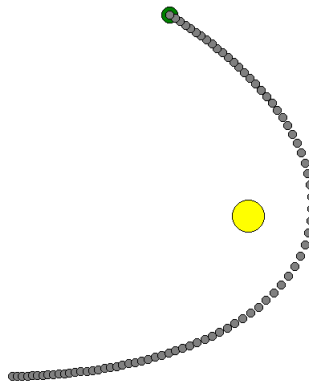


Figure de gauche : la Terre (en bleu) et le Soleil. Le Soleil est fixe. La Terre parcourt une orbite elliptique. Note que lorsqu'elle se rapproche du Soleil la Terre va plus vite.

Figure de droite : Mars (en rouge à l'extérieur), la Terre (en bleu) et le Soleil. Note que le mouvement de la Terre est perturbé par Mars. Dans la réalité le mouvement de la Terre est une ellipse quasiment circulaire et Mars influe peu sur le mouvement de la Terre (car sa masse est considérablement plus faible que celle du Soleil).

On peut aussi tracer la trajectoire d'une comète, ayant une vitesse initiale importante, qui passerait une unique fois près du Soleil avant de s'éloigner définitivement. Elle suit une trajectoire hyperbolique (figure ci-dessous).



Travail à faire :

1. Programme une classe `Planete` héritée de la classe `Particule`. Cette classe contient deux méthodes :
  - une méthode `action_attraction(self, other)` qui calcule la force d'attraction ( $g_x, g_y$ ) entre l'astre courant (l'objet `self`) et un autre astre (l'objet `other`) et qui modifie la vitesse de l'astre courant (de masse  $m_1$ ) selon la formule :

$$\begin{cases} v'_x &= v_x + \frac{g_x}{m_1} \\ v'_y &= v_y + \frac{g_y}{m_1} \end{cases}$$


- une méthode `mouvement(self)` qui ne fait appel qu'à la méthode `action_vitesse()`.
2. Programme une classe `TkPlanete` héritée de la classe `Planete` (de la même façon que `TkParticule()` était héritée de la classe `Particule()`) avec une méthode `affiche()` qui réalise l'affichage graphique avec `tkinter`.
  3. Utilise ton programme pour tracer les orbites de planètes. Par exemple avec trois astres (Soleil, Terre, Mars), à chaque pas il faut calculer l'attraction entre la Terre et le Soleil, l'attraction entre la Terre et Mars, puis déplacer la Terre. Ensuite il faut faire la même chose avec Mars. On considère que le Soleil est fixe.

Voici des exemples de paramètres utilisés (avec en plus  $G = 100$ ).

```
# Trois astres : Soleil, Terre et Mars
soleil = TkPlanete(0,0,0,0,100,"yellow")
terre = TkPlanete(-200,0,0,-5,3,"blue")
mars = TkPlanete(-300,0,0,-5,2,"red")
```

```
for k in range(200):
    terre.action_attraction(soleil)
    terre.action_attraction(mars)
    terre.mouvement()
    terre.affiche(avec_trace=True)
    mars.action_attraction(soleil)
    mars.action_attraction(terre)
    mars.mouvement()
```





```
mars.affiche(avec_trace=True)  
  
canvas.update()  
sleep(0.02)
```