Exercice 1 : Géométrie – Points et Rectangles

1) La classe Point

- Définir une classe *Point*, chaque objet de la classe étant caractérisé par son abscisse et son ordonnée. Ecrire la "méthode constructeur" *__init__* associée, les abscisses et ordonnées par défaut prendront la valeur 0.
- Ecrire une méthode *__repr__* qui affiche les coordonnées d'un point sous la forme :
  coordonnée horizontale = ...
  coordonnées verticale = ...
- Ecrire une méthode *distance* qui **retourne** la distance d'un point à l'origine.
- Ecrire les méthodes **accesseurs** *getx* et *gety* qui permettent de **retourner** respectivement l'abscisse et l'ordonnée d'un objet de la classe *Point.* Ces méthodes seront utilisées par les autres classes afin d'accéder aux attributs abscisse et ordonnée d'un point.
- Ecrire les méthodes **mutateurs** *setx* et *sety* qui permettent de modifier respectivement l'abscisse et l'ordonnée d'un objet de la classe *point.* Ces méthodes seront utilisées par les autres classes lorsqu'elles ont besoin de modifier les attributs abscisse et ordonnée d'un point.
- Définir la méthode *deplace* qui permet de déplacer le point d'une distance dx sur l'axe des abscisses et dy sur l'axe des ordonnées. Cette méthode **modifie** les attributs de l'instance.
- Écrire une méthode **longueur** qui calcule la distance entre 2 points.
- Écrire un programme principal qui calcule le périmètre d'un triangle caractérisé par ses trois sommets A, B et C (par ex. A (0,1), B (7,3), C (2,5))

2) La classe Cercle

- Définir une classe *Cercle*, chaque objet étant caractérisé par son centre (instance de la classe *Point*) et son rayon
- Définir une méthode *dessiner* qui dessine le cercle avec *turtle*
- Ecrire une méthode *ecart* qui retourne l'écart entre deux cercles (= longueur entre les deux centres des cercles)

3) La classe Rectangle

- Définir une classe *Rectangle*, un objet de cette classe est définie par les attributs : *largeur*, *hauteur* et la position du *coin* inférieur gauche qui est un objet de type *Point* qui par défaut sera l'origine de coordonnées (0, 0). (Pour simplifier, nous considérons que ces rectangles seront toujours orientés horizontalement ou verticalement, et jamais en oblique. La *largeur* désigne la longueur du côté horizontal, la *hauteur* celle du côté vertical)
- Ecrire une méthode *__repr__* pour afficher les caractéristiques d'un rectangle.
- Ecrire une méthode *centre* qui calcule le centre du rectangle (le résultat de cette méthode est un **Point**).
- Ecrire une méthode *setCoin* qui modifier le coin d'un rectangle. Cette méthode modifie l'attribut *Coin.*

4) La classe Carre

- Définir la classe *Carre* qui hérite de la classe *Rectangle.* Définir son constructeur *__init__* et redéfinir la méthode *__repr__* qui lui est associée.
- Tester les méthodes *centre, setCoin* sur une instance de la classe *Carre.*

Exercice 2 Complexes

1) Définir la classe *Complexe* qui permet de modéliser un complexe en coordonnées cartésiennes (partie réelle, partie imaginaire).

2) Définir la méthode *__repr__* associée à cette classe.

3) Définir la méthode *norme* qui **retourne** la norme (module) d'un nombre complexe.

4) Définir la méthode *multiplie* qui multiplie un nombre complexe par un nombre réel passé en paramètre. Cette méthode **modifie** les attributs du nombre complexe traité.

5) Définir la méthode *conjugue* qui **retourne** le conjugué d'un nombre complexe. Cette méthode **retourne** un nouveau complexe.

6) Définir la méthode *inverse* qui **retourne** l'inverse d'un nombre complexe. (L'inverse d'un nombre complexe est égal à son conjugue / norme² si la norme n'est pas nulle). Cette méthode **retourne** un nouveau complexe.

7) a) Définir la méthode *__eq__* qui *surcharge* l'opérateur de comparaison "==".

b) Définir la méthode *__add__* qui surcharge l'opérateur d'addition "+".

Exercice 3 : Formes géométriques

On souhaite manipuler diverses formes géométriques : *segments, rectangles, cercles...*
Tous ces objets ont au minimum deux attributs, un ***centre*** (objet de classe Point) et une ***couleur***.

1) Classe Point

Définir :

- une classe *Point*, chaque instance de la classe est caractérisée par ses deux coordonnées *x* et *y*. Ecrire la "méthode constructeur" *__init__* associée, les abscisses et ordonnées par défaut prendront la valeur 0.
- la méthode *__repr__* qui permet d'afficher les attributs des instances de cette classe.
- les méthodes "accesseurs" *getx* et *gety* qui permettent de retourner respectivement l'abscisse et l'ordonnée d'un objet de la classe *point.* Ces méthodes seront utilisées par les autres classes afin d'accéder aux attributs.
- les méthodes "mutateurs" *setx* et s*ety* qui permettent de modifier respectivement l'abscisse et l'ordonnée d'un objet de la classe *point.* Ces méthodes seront utilisées par les autres classes lorsqu'elles ont besoin de modifier les attributs abscisse et ordonnée d'un point.
- une méthode *distance* qui calcule et retourne la distance d'un point à l'origine.
- Définir la méthode *deplace* qui permet de déplacer le point d'une distance *dx* sur l'axe des abscisses et *dy* sur l'axe des ordonnées. Cette méthode modifie les attributs de l'instance.
- Ecrire une méthode *longueur* qui retourne la longueur d'un segment entre deux points
- Écrire un programme principal qui calcule le périmètre d'un triangle caractérisé par ses trois sommets A, B et C (par ex. A (0,1), B (7,3), C (2,5))

2) Classe Forme

Définir :

- la classe *Forme*, et sa méthode *__init__*. Chaque instance de cette classe a deux attributs : un *centre* et une *couleur*. Le centre est une instance de la classe Point. Le centre par défaut est le **point** d'origine ;
- la méthode *__repr__* qui affiche les attributs d'une forme ;
- la méthode *deplace* qui permet de déplacer le centre de la forme d'une distance de *dx* sur l'axe des abscisses et de *dy* sur l'axe des ordonnées ; cette méthode fait bien entendu appel à la méthode *deplace* de la classe *Forme* ;
- la méthode *setCouleur* qui permet de changer l'attribut *couleur* de l'objet.

3) Classe Rectangle

Définir :

- une classe *Rectangle* héritée de la classe *Forme*, un rectangle étant défini par sa largeur et sa hauteur, sa couleur, son centre ;
- la méthode *calculeSurface* qui retourne la surface d'un rectangle ;
- la méthode *dessiner* en **adaptant** la fonction dessiner ci-dessous (disponible sur Arche) qui permet de dessiner un *Rectangle* à l'aide du module *turtle :*

```
def dessiner(xcentre, ycentre, larg, haut, coul):
   """ methode qui dessine un rectangle avec le module turtle """

    up()       # on lève le crayon

    # on se place au debut du trace calculé à partir du centre
    goto(xcentre - larg//2, ycentre - haut//2)
    color(coul)
    begin_fill()
    down()
    setheading(0)

    # demarrage du dessin
    for i in range(2):
        forward(larg)
        right(90)
        forward(haut)
        right(90)
    end_fill()

    up()
```

1ère application

- Créer un point *ctre* de coordonnées 0 et -20
- Créer un rectangle de largeur 90, de hauteur 20, de couleur bleu et de centre le point *ctre* créé précédemment
- Dessiner le rectangle
- Modifier la couleur du rectangle en jaune
- Redessiner le rectangle
- Créer un second rectangle de largeur 45, de hauteur 5, de couleur rouge et de centre le même point créé précédemment
- Dessiner le rectangle
- Déplacer le rectangle1 de 100 sur l'axe des x et de 20 sur l'axe des y
- Dessiner à nouveau des deux rectangles..
- Que remarquez-vous ? Comment faire pour que, bien qu'ayant un centre de mêmes coordonnées au départ, les deux rectangles soient indépendants et qu'on puisse déplacer le centre de l'un sans déplacer le second (il y a plusieurs solutions) ?

4) Classe Cercle

Définir :

- une classe *Cercle* héritée de la classe *Forme*, un cercle étant défini par son diamètre, sa couleur et son centre ;
- la méthode *calculeSurface* qui retourne la surface d'un cercle ;
- la méthode *dessiner*, en adaptant la fonction dessiner ci-après qui permet de dessiner un *Cercle* à l'aide du module *turtle :*

```
def dessiner(xcentre, ycentre, rayon, coul):
    """ methode qui dessine un cercle avec le module turtle """

    up()    # on lève le crayon

    # placement au debut du dessin calculé par rapport au centre
    goto(xcentre, ycentre - rayon)
    color(coul)
    begin_fill()
    down()
    # dessin du cercle
    circle(rayon)
    end_fill()
    up()
```

5) Application

- Créer la liste de formes suivante :

  lformes = [Cercle(100, "yellow"), Cercle(20, "black", Point(25,12)), Cercle(20, "black", Point(-25, 12)), Rectangle(45, 5, "red", Point(0,-20))]

- Ecrire l'itération qui permet de dessiner chacune des formes de cette liste
- Ecrire l'itération qui déplace chaque forme de 50 unités vers la droite et de 100 unités vers le haut.
- Relancer l'itération qui dessine chacune des formes.

Exercice 4 : Photos

1) Définir une classe *Date* :

- le constructeur reçoit en paramètre le jour (jj), le mois (mm) et l'année (aaaa) de la date à créer, *par exemple : 25, 1 et 1989*
- définir la méthode qui permet d'afficher une date sous la forme "jj mois aaaa", *par exemple "25 janvier 1989"*.

2) Définir une classe *Support* qui permet de modéliser les supports numériques tels que les écrans, les imprimantes, les scanners, etc. :

- le constructeur reçoit en paramètre :
  - un nom
  - une résolution : elle détermine le nombre de pixels par unité de longueur (*dpi, dots per inch* ou *ppp, points par pouces*, un pouce vaut à peu près 2,54 cm).

3) Définir une classe *Photo* :

- le constructeur reçoit en paramètre le nom de l'auteur de la photo, la date de prise de vue, les dimensions de la photo (en pixels), une liste de mots-clés la caractérisant et la "licence" d'utilisation de la photo, par défaut « CC BY-NC-SA »[1].
- Définir la méthode *calculer_dimensions* qui calcule et **retourne** la dimension d'une photo en cm en fonction du **support** qui la reçoit. Cette dimension sera calculée à partir de la résolution du support sur lequel elle sera affichée ou imprimée[2] [3].
- Définir deux sous-classes de la classe *Photo* :
  - La sous-classe *Photo_Numérique* : une photo au format numérique a toujours (ou presque) le même rapport Hauteur / Largeur qui est de 3/4[4]. Pour créer une photo numérique, il suffit donc de donner la hauteur de la photo, en nombre de pixels, la largeur sera calculée dans le constucteur *__init__*.
  - La sous-classe *Photo_Traditionnelle* : une photo au format traditionnel a en revanche un rapport Hauteur / Largeur de 2/3. Pour créer une photo au format traditionnel, il suffit donc de donner la hauteur de la photo, la largeur sera calculée dans le constucteur *__init__*.

4) Définir une classe *Tirage_photos.*

- Les instances de cette classe sont créées à partir des dimensions en cm d'un tirage photo et d'une photo.
- Définir une méthode *qualite* qui évalue la qualité du tirage photo en fonction de la résolution de la photo et de la taille du tirage souhaité. On prendra les critères

1 http://eduscol.education.fr/internet-responsable/se-documenter-publier/produire-et-publier-ses-propres-contenus/diffuser-des-contenus-reutilisables-par-dautres.html

2 http://sebsauvage.net/comprendre/dpi/index.html

3 http://www.imedias.pro/cours-en-ligne/graphisme-design/definition-resolution-taille-image/la-resolution-pour-une-image/

4 http://www.photograpix.fr/blog/trucs-et-astuces/quel-format-photo-choisir-et-comment-modifier/

suivants, fonction de la résolution du tirage, calculé à partir de la définition de la photo et de la taille du tirage [1] :

 Excellent > 300 dpi

 Correct > 200 dpi

 Déconseillé < 200 dpi

Jeu d'essais :

Tests pour les photos

A tester pour les supports :

- écran à 94 dpi
- imprimante à 300 dpi
- photo : définition 3872x2592
  - taille à obtenir pour l'écran : environ 70 cm x 104,2 cm
  - taille à obtenir pour l'imprimante : environ 22 cm x 32,8 cm
- photo : définition 800x600
  - taille à obtenir pour l'écran : environ 16.21 cm x 21.62 cm
  - taille à obtenir pour l'imprimante : environ 5.08 cm x 6.77 cm

Tests pour les tirages

- photo numérique de hauteur 1536
  - tirage de dimension 13x10 (qualité Excellent)
  - et tirage de dimension 18x13 (qualité Correct)
- photo traditionnelle de hauteur 400
  - tirage de dimension 11x15 (qualité Déconseillé)

Exercice 5 : Atomes

Les ***atomes*** sont constitués d'un certain nombre de ***protons*** (particules chargées d'électricité positive), d'***électrons*** (chargés négativement) et de ***neutrons*** (neutres).

Le type d'atome (ou élément) est déterminé par le nombre de protons, que l'on appelle également ***numéro atomique***. Dans son état fondamental, un atome contient autant d'électrons que de protons, et par conséquent il est électriquement neutre. Il possède également un nombre variable de neutrons (variable suivant les isotopes de chaque atome). Dans certaines circonstances, un atome peut gagner ou perdre des électrons. Il acquiert de ce fait une charge électrique globale, et devient alors un ***ion*** (il s'agit d'un ***ion négatif*** si l'atome a gagné un ou plusieurs électrons, et d'un ***ion positif*** s'il en a perdu). La charge électrique d'un ion est égale à la différence entre le nombre de protons et le nombre d'électrons qu'il contient.

1) La classe *Atome*

[1] http://www.photobox.fr/contenu/resolution-photo

a) Définir une classe *Atome* :

- Le constructeur reçoit en paramètre le nom de l'atome, le numéro atomique et le nombre de neutrons (0 par défaut).
- Chaque objet de cette classe a comme attributs un nom, un nombre de protons (déterminé par le numéro atomique), un nombre d'électrons identique au nombre de protons, un nombre de neutrons.

b) Ecrire la méthode *__repr__* qui permet d'afficher les attributs d'une instance de cette classe.

c) Définir les méthodes accesseurs *getNom*, *getNbProtons*, *getNbNeutrons* qui retourne respectivement la valeur des attributs *Nom*, *Protons*, *Neutrons*.

d) Ecrire la méthode qui calcule la masse d'un atome. La masse d'un atome est la somme de la masse des électrons, de la masse des protons et de la masse des neutrons :

$\mathbf{m}_{\mathbf{protons}} = 1.,67265 \times 10^{-27}$ kg
$\mathbf{m}_{\mathbf{neutrons}} = 1.67496 \times 10^{-27}$ kg
$\mathbf{m}_{\mathbf{electrons}} = 9.10953 \times 10^{-31}$ kg

Application :

a) Créer un objet modélisant l'atome d'or. Calculer la masse d'un atome d'or (nombre de protons = 79 ; nombre de neutrons = 118). La plus grosse pépite d'or trouvée en France a une masse **m** = 543 g. Combien d'atomes d'or contient cette pépite ?

b) Créer un dictionnaire correspondant aux 5 premiers éléments de la table des éléments périodiques sans tenir compte du nombre de neutrons (sinon vous référer aux sources suivantes pour prendre les isotopes les plus stables[2]). La clé de chaque élément est son symbole chimique, sa valeur est un objet de type *Atome* modélisant l'élément atomique correspondant.

c) Ecrire un programme qui grâce à la table créée dans la question précédente recherche le nom de l'atome possédant 4 protons.

[2] http://fr.wikipedia.org/wiki/Isotope#stable

2) La classe *Ion*

a) Définir une classe *Ion* qui dérive de la classe *Atome*, mais qui possède en plus la charge qui détermine chaque objet de cette classe. Pour instancier des objets *Ion*, on doit fournir un nom, un numéro atomique, une charge électrique globale (positive ou négative) et un nombre de neutrons (0 par défaut) qui permet de calculer le nouveau nombre d'électrons.

b) Ecrire la méthode *__repr__* qui permet d'afficher les attributs associés à un objet de cette classe. Cette méthode pourra faire appel à la méthode d'affichage associée à la classe *Atome*.

Applications

L'ion fluorure présent dans les dentifrices au fluor est un atome de fluor (symbole atomique F, numéro Atomique 9, nombre de neutrons 10) qui a gagné un électron. Créer un objet qui modélise l'ion fluor. Quelle est sa masse ?

Exercice 6 : Gestion de la vente en ligne de produits culturels

On demande de modéliser la gestion d'un site de vente en ligne.
Ce site propose à la vente des ouvrages : livres, CD et DVD.
Chaque ouvrage est caractérisé par :

- une référence
- un titre
- un prix hors taxe
- un stock

Les livres sont de plus caractérisés par le nom de l'auteur, les CD par le nom du chanteur et le nombre de plages, les DVD par le nom du réalisateur et sa durée.
On souhaite écrire les méthodes suivantes :

- calculePrixTTC :
  - qui calcule le prix TTC prenant en compte la TVA ;
  - le taux de TVA est de 19,6% sauf pour les livres pour lesquels il est de 7%
- calculeFraisEnvoi :
  - pour les livres, les frais d'envoi s'élèvent à 2 € si le livre pèse moins de 100g et à 4,07 € sinon ;
    - le poids du livre est calculé à partir du nombre de pages ;
    - le poids d'une page est évalué à 5g et le poids de la couverture à 15g ;
    - un livre est donc également caractérisé par le nombre de pages qui le composent.
  - pour les produits multimédias (CD et DVD), les frais d'envoi s'élèvent à 2,5 € s'il n'y a qu'un seul support (un seul CD ou un seul DVD) et 3,5 € sinon
- réapprovisionnement :
  - qui retourne vrai si on doit lancer un réapprovisionnement, faux sinon ;
  - pour les livres, le réapprovisionnement est nécessaire si le stock est <= 3
  - pour les CD, le réapprovisionnement est nécessaire si le stock est <= 2
  - pour les DVD, le réapprovisionnement est nécessaire si le stock est <= 1

Proposer une représentation sous forme de classes qui permettent la création de ces différents types d'objets ainsi que la définition des méthodes décrites ci-dessus, en limitant au maximum les redondances.

Exercice 7 : La chenille en Objets

Programmer à nouveau l'exercice sur le déplacement d'une chenille en mettant en œuvre la Programmation orientée objet. Dans ce programme vous utiliserez une classe *Chenille* et une classe *Anneau.*

Les instances de la classe *Chenille* auront comme attributs : le nombre d'anneaux, la position de départ de la chenille, la couleur des anneaux et le rayon des anneaux.

Les instances de la classe *Anneau* auront comme attributs un rayon, une couleur et une position.

Ajouter une sous-classe *Tete* de la classe *Anneau* telle que la tête est toujours de couleur grise et lorsqu'on la dessine, la circonférence est marquée par un trait plus épais.