Il faut avoir lu le document d'introduction et traiter la conception de la classe Cercle pour aborder ce document.

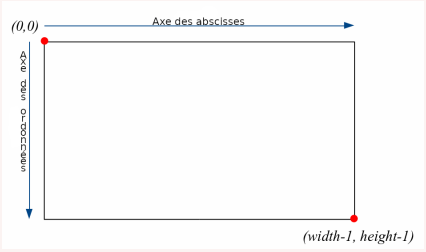
On trouvera sur le dépôt `fond_ecran` du github de la classe le fichier `fond_ecran.py` partiellement complété. Ce fichier prend en compte que la classe `Cercle` a été enregistré dans un fichier nommé `cercle_cairo.py`.

1. Vérifier que votre nom de fichier contenant la classe `Cercle` se nomme bien `cercle_cairo.py`
2. Dans cette question, on s'intéresse au constructeur déjà codé. À partir de la documentation officielle de `pycairo` :
   a) Expliquer ce qu'est le format `cairo.FORMAT_ARGB32`
   b) Pourquoi, compte-tenu des signatures proposés, aurait-on pu choisir un autre format ? Si oui, le(s)quel(s) ?
   c) Que fait la méthode `paint` ?
   d) Pourquoi y a-t-il un astérisque dans `self.ctx.set_source_rgb(*couleur)` ?
3. Dans cette question, on s'intéresse à la méthode `est_dans_un_cercle(point)`.
   Il s'agit de vérifier que le point passé en argument peut devenir centre d'un cercle : s'il est dans un disque déjà tracé, ce ne sera pas le cas. Pour chacun des cercles tracés, il faut donc tester la distance du centre avec ce point-là (et donc utiliser une méthode de la classe `Cercle`).

   Compléter le code fourni en relisant bien le rôle de cette méthode.

   Rappel : `est_dans_un_cercle(point: Tuple[int, int])⟶bool`
4. Dans cette question, on s'intéresse à la méthode `rayon_max_bords(position)`

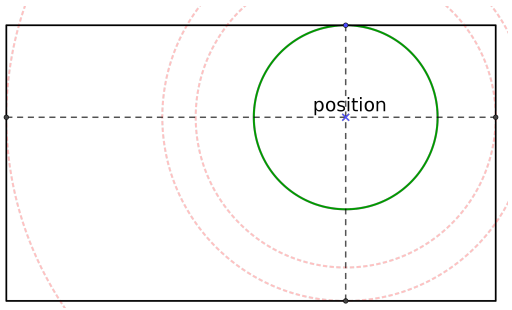
Important



Les zones de dessin, aussi bien dans PIL(low) que dans `pycairo` ne sont pas orientés comme le plan euclidien (mathématique) usuel : l'origine est en haut à gauche et l'axe des ordonnées est orienté vers le bas.

Il y a 4 cas possibles de débordement, correspondant à chacun des côtés de l'image.

Dans la figure ci-contre, le rayon maximal possible sans déborder est acquis avec le bord haut, les autres cercles (en pointillés) débordent de l'image.
On calcule donc les distances aux quatre bords et le rayon à choisir est la distance la plus petite.

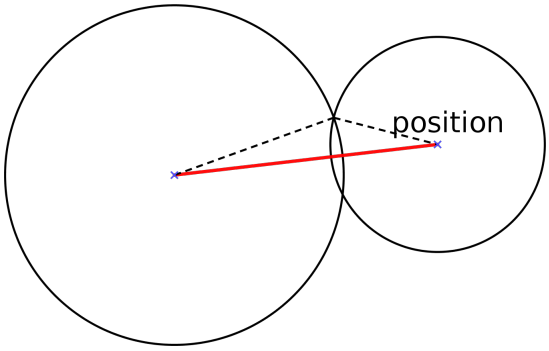


Compléter le code fourni en relisant bien le rôle de cette méthode.

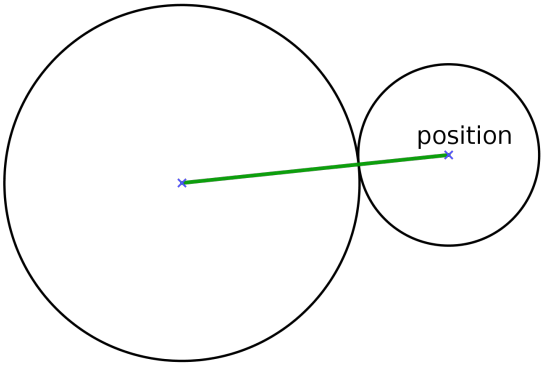
Rappel : `rayon_max_bords(position: Tuple[int, int])⟶int`

5. Dans cette question, on s'intéresse à la méthode `rayon_max_cercles(position)`

   Observons d'abord ces deux figures :

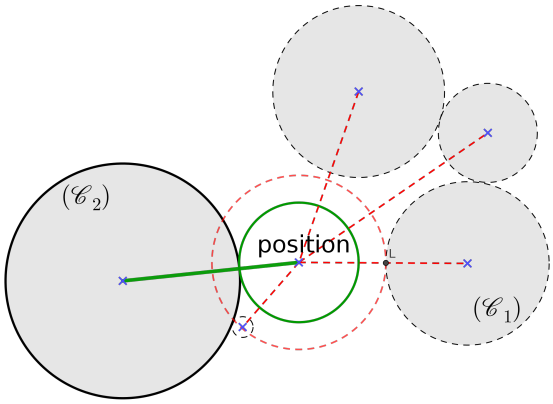


Dans la situation ci-dessus, la somme des rayons est supérieure à la distance entre les deux centres : il y a chevauchement.



Dans la situation ci-dessus, la somme des rayons est égale à la distance entre les deux centres : il n'y a pas chevauchement.

Dans la figure ci-dessous, on étudie le calcul d'un rayon selon la position :



- `position` : la position passée en argument
- Disques colorés : cercles déjà présents
- Cercle pointillé : cercle possible avec $(\mathscr{C}_1)$
- Traits pointillés : segments de distance aux centres des disques déjà tracés
- Cercle en trait plein : cercle définitif, il ne chevauche aucun des cercles déjà tracés et est le plus grand possible (il est associé à $(\mathscr{C}_2)$)

Pour trouver le rayon du cercle définitif

Pour chacun des cercles représentés sous forme de disques :

- on **calcule la distance** de son centre avec la position ;
- on **enlève le rayon du cercle** : on obtient alors le rayon qui permet au deux cercles de se toucher ;
- on le sélectionne s'il est plus petit que le rayon candidat précédent.

La fonction doit donc réaliser une boucle sur l'ensemble des objets cercles déjà dessinés (c'est l'attribut `cercles` qui maintient une liste), calculer une distance (voir précédemment) et choisir pour rayon le plus petit nombre entre cette distance et le rayon actuel.

Compléter le code fourni en relisant bien le rôle de cette méthode.

Rappel : `rayon_max_cercles(position: Tuple[int, int])⟶int`

6. Dans cette question, on s'intéresse à la méthode `tracer_cercles(effectif, couleur, max_admis)`

L'algorithme pour tracer les cercles

a) Choix d'une position au hasard dans l'image.

b) Vérification si cette position convient, sinon retour au point a).

c) Calcul du rayon maximum qu'on peut mettre à cette position, en tenant compte d'un maximum admissible.

d) Tracer du nouveau cercle.

e) Ajout de ce nouveau cercle à la liste des cercles déjà tracés.

On itère ce processus jusqu'à obtenir le bon effectif de cercles.

a) N'y a-t-il pas un GROS problème dans cet algorithme ? (problème qu'on ne réglera pas).

b) Compléter le code fourni en relisant bien le rôle de cette méthode.

Rappel :

```
tracer_cercles(effectif: int, couleur: Tuple(float, float, float), max_admis: int)⟶None
```