Ciel 1- Script d'une fonction qui calcule *cos* et *sin*

1- TRACE DU CERCLE TRIGONOMETRIQUE EN UTILISANT LA BIBLIOTHEQUE *TURTLE*

Ce travail est noté. On demande de rédiger un compte-rendu au format *.doc* ou *.odt* . qui contiendra :

- les réponses aux différentes questions posées,
- les captures d'écran **des morceaux de codes** écrits **et** celles **des résultats des exécutions**. Pour faire ces captures, utiliser *l'Outil Capture d'écran* de Windows (touches clavier *windows+Shift+s*)

a. BIBLIOTHEQUE *TURTLE*

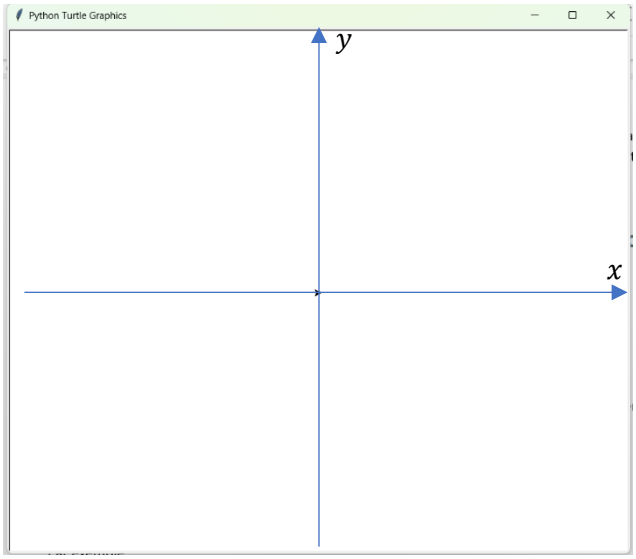
La bibliothèque *turtle* de python, permet de créer facilement une fenêtre graphique et d'y tracer des traits par déplacement d'un crayon appelé « *tortue* ». Les fonctions proposées dans cette bibliothèque sont présentées sur ce lien : https://docs.python.org/fr/3/library/turtle.html

Au démarrage du script, il faut importer la bibliothèque *turtle*.

En exécutant le script ci-contre, une fenêtre graphique s'ouvre. La fonction *mainloop()* permet de maintenir la fenêtre graphique ouverte.

```
1 from turtle import *
2
3
4 mainloop()
```

En intercalant des fonctions python de *turtle* entre la 1ère et la dernière ligne, une « tortue », représentée par ce symbole ➤, peut se déplacer et laisser sur la fenêtre la trace de ses mouvements.

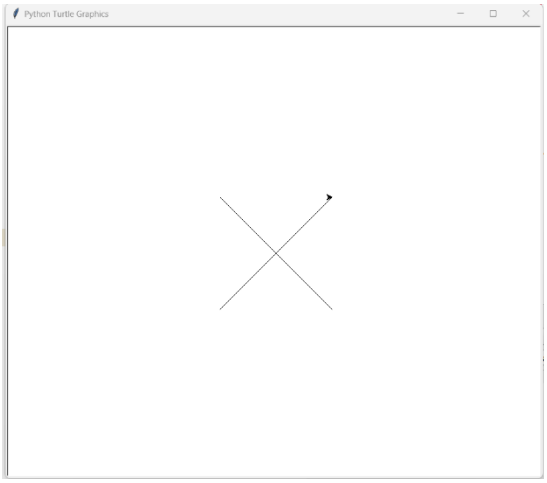


Au démarrage du script, la tortue se trouve sur l'origine d'un repère (x, y) centré dans la fenêtre. L'unité est le pixel.

Par exemple, le script ci-dessous :

```
1  from turtle import *
2
3  up()
4  goto(100,-100)
5  down()
6  goto(-100,100)
7  up()
8  goto(-100,-100)
9  down()
10 goto(100,100)
11
12
13 mainloop()
```

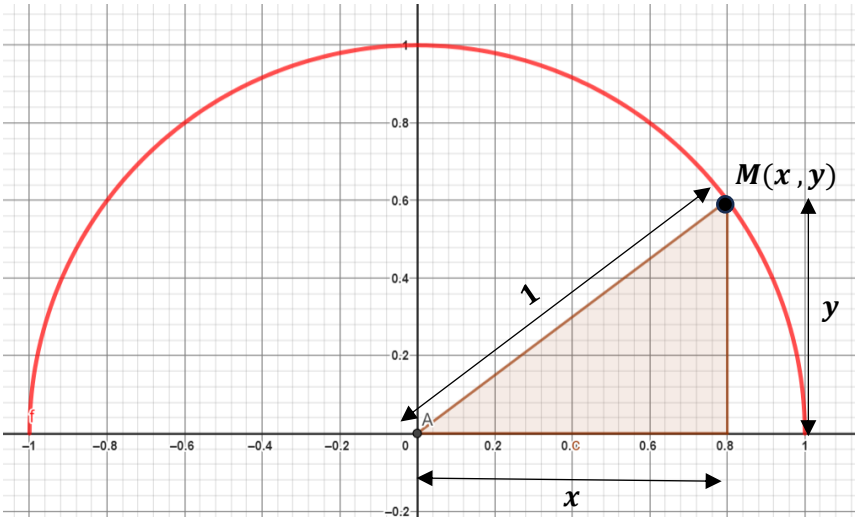
permet d'obtenir :



Q1. Ecrire de la même façon un script qui permet de tracer un carré de coté 400 px et centré dans la fenêtre graphique.

b. FONCTION DONT LA COURBE EST UN DEMI-CERCLE DE RAYON 1

Mathématiquement, la courbe représentative d'une fonction f est un ensemble de points M de coordonnées (x, y) tels que $y = f(x)$.



Cette courbe est le cercle trigonométrique, c'est-à-dire le cercle de rayon 1 et centré sur l'origine du repère, si on a la relation suivante entre x et y :

$$x^2 + y^2 = 1^2$$

De cette relation, obtenue en appliquant le théorème de Pythagore sur le triangle repéré ci-dessus, on peut obtenir :

$$y^2 = 1 - x^2$$

Et donc : $y = \sqrt{1 - x^2}$ ou $y = -\sqrt{1 - x^2}$

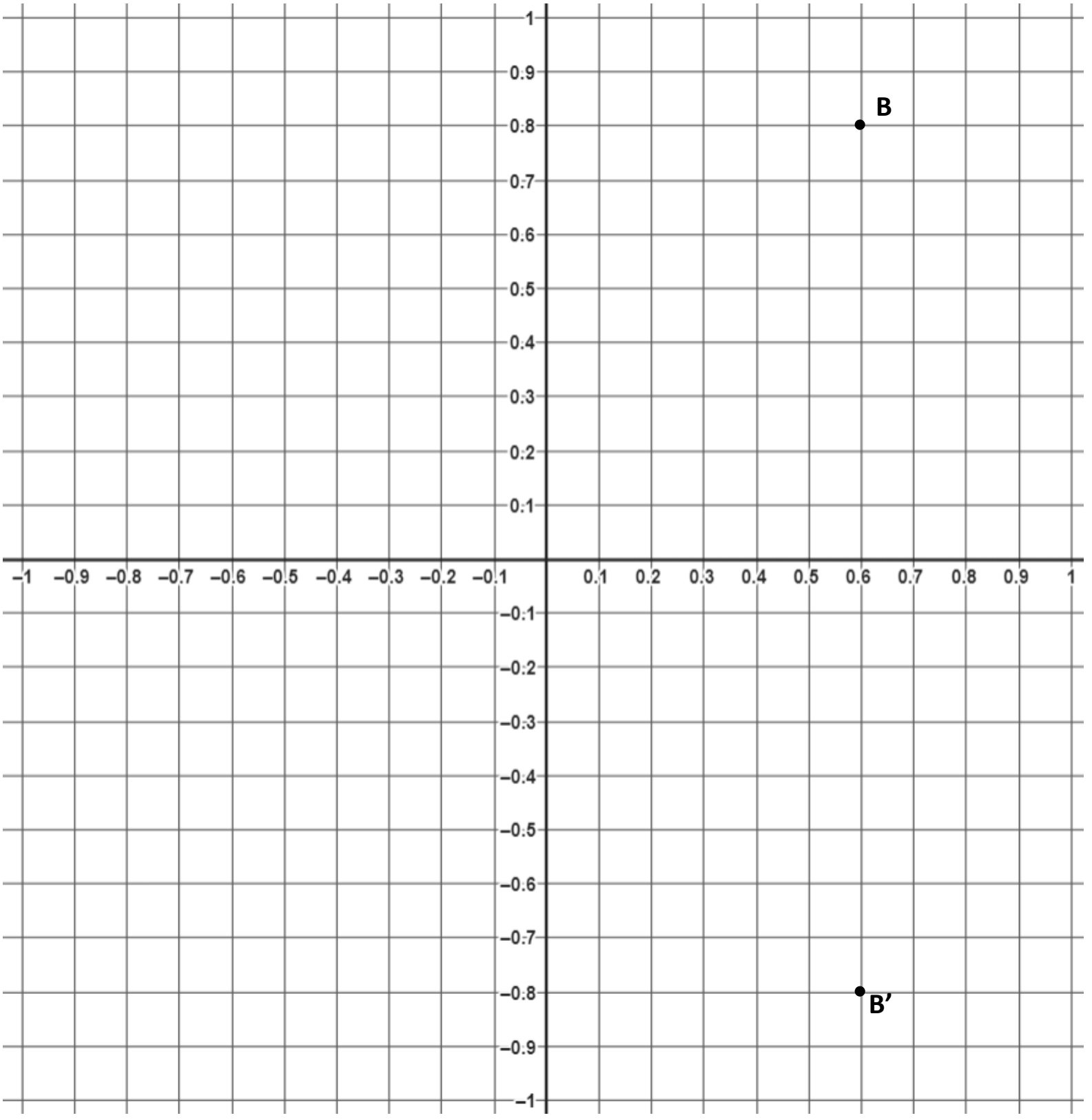
Ainsi la fonction f définie pour $-1 \leq x \leq 1$ par $f(x) = \sqrt{1 - x^2}$ aura comme courbe représentative le demi-cercle trigonométrique supérieur.

La fonction définie pour $-1 \leq x \leq 1$ par $g(x) = -\sqrt{1 - x^2}$ permet d'obtenir le demi-cercle inférieur.

Q2. Compléter le tableau ci-dessous sur le document réponse donné, en calculant les ordonnées des points A, C, D, E, F, A', C', D', E' et F' .

x	1	$\mathbf{0,6}$	0,2	-0,2	-0,6	-1
$y = \sqrt{1 - x^2}$		$\sqrt{1 - \mathbf{0,6}^2} = 0,8$				
Points cercle supérieur	$A(1\ ;\ \ )$	$B(0,6\ ;0,8)$	$C(0,2\ ;\ \ )$	$D(-0,2;\ \ )$	$E(-0,6;\ \ )$	$F(-1\ ;\ \ )$
$y = -\sqrt{1 - x^2}$		$-0,8$				
Points cercle inférieur	$A'(1\ ;\ \ )$	$B'(0,6\ ;0,8)$	$C'(0,2\ ;\ \ )$	$D'(-0,2;\ \ )$	$E'(-0,6;\ \ )$	$F'(-1;\ \ )$

Q3. Positionner les points A, C, D, E, F, A', C', D', E' et F' dans le repère donné sur la page qui suit et sur le document réponse. Tracer les segments $[AB]$, $[BC]$, $[CD]$, $[DE]$, $[EF]$ et $[A'B']$, $[B'C']$, $[C'D']$, $[D'E']$, $[E'F']$.



En traçant ces segments, on est encore loin d’avoir un cercle. Par contre, si le nombre de points est bien plus important, le tracé de ces segments permettra d’obtenir une courbe proche de celle du cercle trigonométrique.

Réaliser cette opération « à la main » étant bien trop fastidieux, on se fixe à présent l’objectif de créer un script python qui permette de le faire.

c. SCRIPT QUI PERMET DE TRACER AVEC TURTLE LE CERCLE TRIGONOMETRIQUE

On commence le script par :

```
1  from turtle import *
2  from math import sqrt
3  Screen().setworldcoordinates(-1 , -1 , 1 , 1)
4
5
6
7
8  #main
9
10 mainloop()
```

On importe la fonction racine carrée de la bibliothèque *math*

On écrira dans cette partie les scripts des fonctions qui seront écrites dans la suite.

On exécute cette méthode pour faire en sorte que le coin bas-gauche de la fenêtre soit sur le point de coordonnées $(-1,-1)$ et le coin haut-droit sur le point de coordonnées $(1\,,1)$

On écrira dans cette partie les lignes pythons qui exécuteront les fonctions

⇨ Enregistrer ce script dans un fichier avec un nom quelconque, dans votre zone de travail sur U: . Pour ne pas provoquer d'erreurs avec Turtle, exécuter à chaque fois le script en allant dans l'onglet *Exécuter* et choisir *Démarrer le script*.

⇨ Pour définir la fonction f définie par $f(x)=\sqrt{1-x^2}$, on écrit le script suivant dans le fichier précédent :

```
def f(x) :
    return sqrt(1-x**2)
```

```
>>> f(0.6)
0.8

>>> f(0.2)
0.9797958971132712

>>> f(-0.2)
0.9797958971132712

>>> f(-1)
0.0
```

⇨ On demande à python de lire le fichier en cliquant sur *Exécuter/Démarrer le script*

⇨ On teste aussitôt le script écrit pour vérifier qu'il donne bien les résultats attendus. On peut, par exemple, **dans la console**, exécuter cette fonction pour les abscisses des points B, C, D, F et vérifier que les valeurs retournées sont bien identiques à celles calculées précédemment « à la main ».

Q4. Ecrire dans le fichier le script qui définit la fonction g définie par $g(x)=-\sqrt{1-x^2}$. Réaliser dans la console 4 tests qui valident l'exactitude de ce script. Ne pas oublier de coller les copies d'écran correspondantes, dans votre compte-rendu.

⇨ Pour tracer un nombre N de segments pour représenter le demi-cercle supérieur, on écrit le script ci-dessous, toujours dans le même fichier.

```
def cercleSup(N) :
    # positionnement du crayon sur le point de coordonnées (1,0)
    up()
    goto(1,0)
    down()
    # bouclage pour le tracé de N segments
    dx = 2 / N  # on divise le diamètre qui est de 2 par N
    for i in range(N+1) :
        x = 1 - i*dx
        y = f(x)
        goto(x,y)
```

⇨ Tester la validité de ce script en exécutant cette fonction dans la partie programme principale (*main*) en fin de fichier, avec 5 segments pour commencer, puis essayer 500 segments.

```
#main
cercleSup(5)
mainloop()
```

⇨ Pour encore mieux vérifier que tout fonctionne correctement, on peut insérer dans la boucle de la fonction, des lignes qui permettent d'afficher la valeurs des coordonnées des points :

```
def cercleSup(N) :
    # positionnement du crayon sur le point de coordonnées (1,0)
    up()
    goto(1,0)
    down()
    # bouclage pour le tracé de N segments
    dx = 2 / N  # on divise le diamètre qui est de 2 par N
    for i in range(N+1) :
        x = 1 - i*dx
        y = f(x)
        goto(x,y)
        print(f"x = {round(x,2)} , y = {round(y,2)} ")
        write(f"x = {round(x,2)} , y = {round(y,2)} ")
```

Affiche dans la **console** les valeurs arrondies de x et y. NB : le *f* (*f* comme *format)* devant les guillemets n'a rien à voir avec la fonction f.

Affiche les mêmes valeurs dans la fenêtre graphique à l'endroit où se trouve le pointeur à ce moment là.

⇨ Tester ce dernier script.

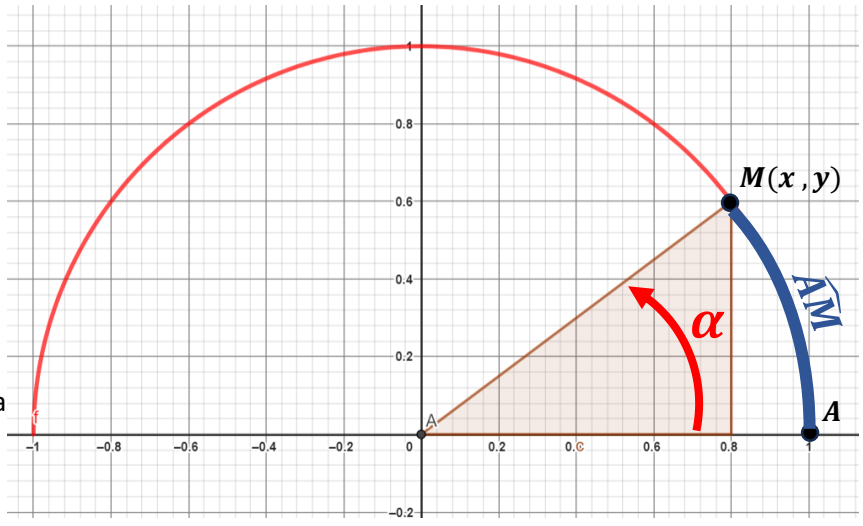
Q5. Ecrire à présent dans le fichier, le script de la fonction *cercleInf()* qui prend en argument un nombre N de segments et permet de tracer le demi-cercle inférieur. Tester l'ensemble, en traçant un cercle complet pour N = 100. Ne pas oublier de coller les copies d'écran correspondantes dans le compte-rendu.

2- CALCUL DE L'ANGLE EN RADIANS POUR CHACUN DES POINTS DU CERCLE.

Pour un point M de coordonnées (x, y) , l'angle α en radian est égal à la longueur curviligne $\widehat{AM}$ entre les points A et M.

Pour calculer cette longueur, il suffit d'ajouter les longueurs de tous les segments qui se trouvent entre le point A et le point M. Si les segments sont de petites tailles, cette méthode nous donnera un résultat proche de la valeur exacte.

On se propose à présent d'écrire un script qui permette de calculer cet angle α en radian.



a. SCRIPT QUI CALCULE LA LONGUEUR D'UN SEGMENT

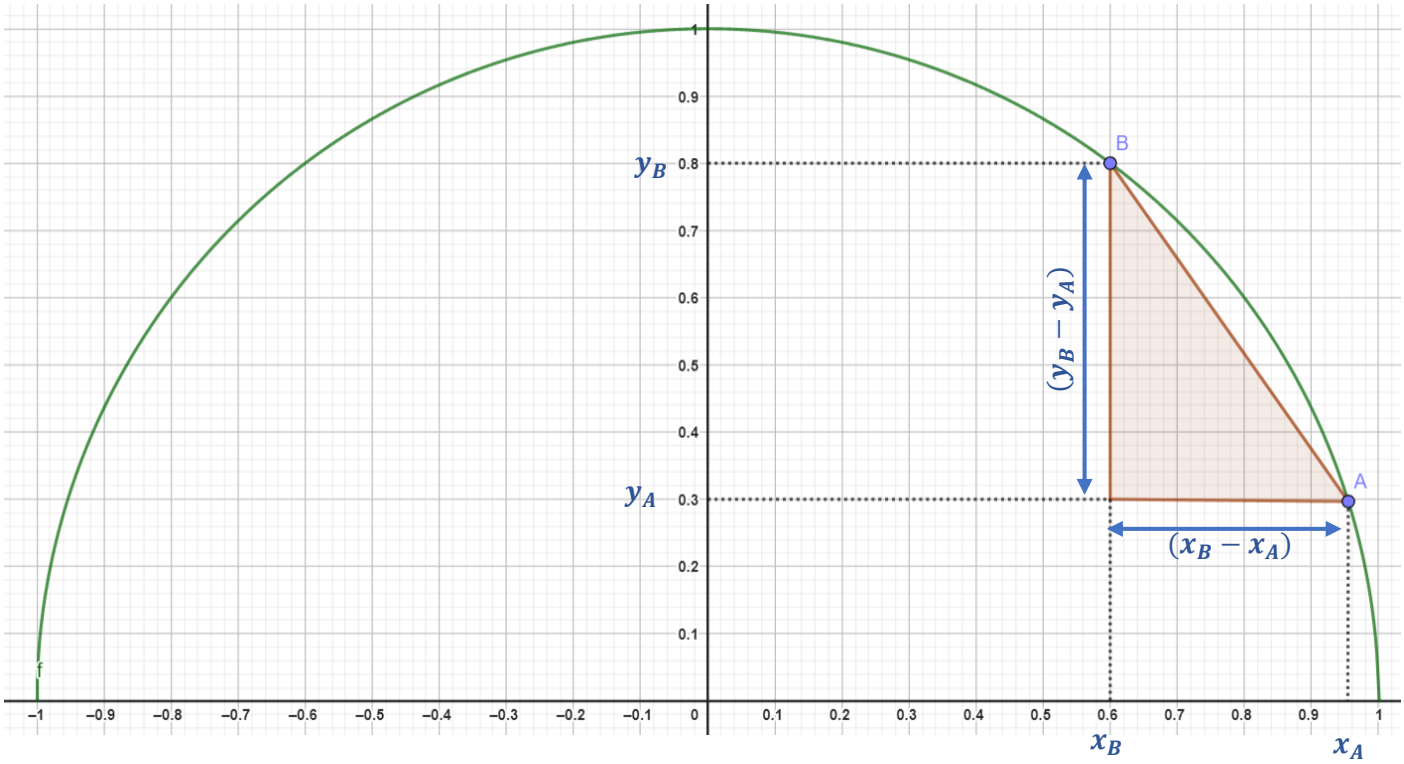
Lorsque l'on connait les coordonnées de 2 points A et B dans un repère orthonormé (repère avec des axes perpendiculaires et avec la même graduation en abscisse et en ordonnée), on peut facilement déterminer la longueur AB en utilisant le théorème de Pythagore sur le triangle rectangle de la figure ci-dessous.

Pour les points $A(x_A\ ;\ y_A)$ et $B(x_B\ ;\ y_B)$, la longueur du segment AB est telle que :

$$AB^2 = (x_B - x_A)^2 + (y_B - y_A)^2$$

On a donc :

$$AB = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$



⇨ Ecrire toujours dans le même fichier, le script de la fonction $longueur()$ qui prend en argument les abscisses de chacun de 2 points du cercle supérieur et retourne la longueur du segment entre ces 2 points :

```
def longueur(xA,xB) :
    yA = f(xA)
    yB = f(xB)
    AB = sqrt((xB-xA)**2 + (yB-yA)**2)
    return AB
```

On teste aussitôt ce script, en demandant tout d'abord à python de lire le fichier *(… Exécuter / Démarrer le script)* pour ensuite pouvoir tester cette nouvelle fonction dans la console. Par exemple, si on prend l'exemple des points B et C définis sur la page 2, et de coordonnées respectivement 0.6 et 0.2 , on exécute dans la console :

```
>>> longueur(0.6,0.2)
0.43855052687092505
```

Ce résultat est-il exact ? … il suffit de le vérifier en mesurant, avec une règle, la longueur de ce segment $[BC]$, tracé sur le document réponse.

b. MODIFICATION DU SCRIPT DES FONCTIONS CERCLESUP( ) ET CERCLEINF( ) :

⇨ On modifie le script de la fonction *cercleSup( )* qui permettait de tracer le demi-cercle supérieur. On se propose d'y ajouter le calcul de l'angle en radians pour chacun des points tracés. On obtient ainsi le script suivant :

```
def cercleSup(N) :
    # positionnement du crayon sur le point de coordonnées (1,0)
    up()
    goto(1,0)
    down()
    # bouclage pour le tracé de N segments
    dx = 2 / N  # on divise le diamètre qui est de 2 par N
    angle = 0
    for i in range(N+1) :
        x = 1 - i*dx
        y = f(x)
        goto(x,y)
        if i != 0 :
            xPrecedent = 1 - (i-1)*dx
            angle = angle + longueur(x , xPrecedent)
            print(f"x={round(x,2)} y={round(y,2)} a={round(angle,2)} rad")
            write(f"a={round(angle,2)}")
```

Initialisation de la variable *angle*

La valeur de cette variable *angle* est augmentée à chaque fois

Pour le 1er point, on ne fait rien, car il n'y a pas encore de segments avant. Le symbole *!=* teste la différence

On vérifie que tout fonctionne bien e dans la console n affichant les valeurs des

```
#main
cercleSup(10)

mainloop()
```

On teste aussitôt ce script en commençant par exemple, avec un demi-cercle composé de 10 segments (on exécute dans la partie programme principale : `cercleSup(10)`)

On peut modifier les termes employés pour l'affichage, sachant que le cosinus est égal à l'abscisse du point sur le cercle trigonométrique et le sinus est égal à son ordonnée. Cela peut donner alors :

```
def cercleSup(N) :
    # positionnement du crayon sur le point de coordonnées (1,0)
    up()
    goto(1,0)
    down()
    # bouclage pour le tracé de N segments
    dx = 2 / N  # on divise le diamètre qui est de 2 par N
    angle = 0
    for i in range(N+1) :
        x = 1 - i*dx
        y = f(x)
        goto(x,y)
        if i != 0 :
            xPrecedent = 1 - (i-1)*dx
            angle = angle + longueur(x , xPrecedent)
            print(f"cos({round(angle,2)})={round(x,2)}  sin({round(angle,2)})={round(y,2)}")
            write(f"a={round(angle,2)}")
```

Q6. Modifier à présent de la même façon, le script de la fonction *cercleInf()* . Tester l'ensemble, en traçant un cercle complet pour N = 100. Ne pas oublier de coller les copies d'écran correspondantes dans le compte-rendu.

3- SCRIPT D'UNE FONCTION QUI CALCULE LA MESURE PRINCIPALE D'UN ANGLE EN RADIANS

On rappelle que la mesure principale d'un angle permet de repérer le même point sur le cercle trigonométrique, mais avec un angle α tel que $-\pi < \alpha \leq \pi$.

Par exemple pour calculer la mesure principale de l'angle 2023 rad, on détermine « le nombre de tours qu'il y a dans la valeur 2023 » : $\frac{2023}{2\pi} \approx 321{,}97$

On retient l'entier le plus proche : $round(321.97\,,0) = 322$

Pour avoir la mesure principale α de 2023 rad, on enlève l'équivalent en radians, de 322 tours en radians :

$$\alpha = 2023 - 2\pi \times 322 \approx -0.1856689\, rad$$

Autre exemple, pour un angle de 2024 rad : $\frac{2024}{2\pi} \approx 322{,}13$

On retient l'entier le plus proche : $round(322.13\,,0) = 322$

Pour avoir la mesure principale α de 2024 rad, on enlève l'équivalent de 322 tours en radians :

$$\alpha = 2024 - 2\pi \times 322 \approx 0.8143311\, rad$$

Q7. Ecrire le script d'une fonction nommée *mesurePrincipale( )* qui prend en argument un angle en radians et retourne sa mesure principale. Par exemple, en exécutant cette fonction dans la console avec les angles de 2023 rad et 2024 rad, on aura :

```
>>> mesurePrincipale(2023)
-0.18566891182672407

>>> mesurePrincipale(2024)
0.8143310881732759
```

NB : Pour avoir le nombre π avec précision, rajouter sur la ligne d'importation de la bibliothèque *math,* pi :

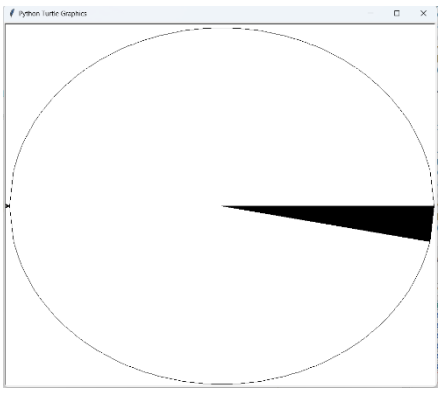
```
from math import sqrt,pi
```

4- SCRIPT D'UNE FONCTION QUI CALCUL LE COSINUS ET LE SINUS D'UN ANGLE

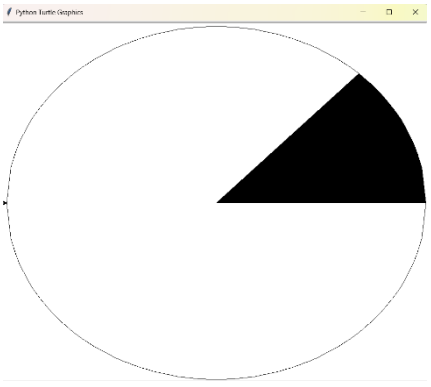
Q8. Ecrire le script d'une fonction nommée *cosinus( )* qui prend en argument un angle en radians et retourne son cosinus en affichant en plus dans une fenêtre graphique, le secteur angulaire noirci de sa mesure principale. Bien sûr, ne pas utiliser la fonction cos( ) disponible dans la bibliothèque *math*, mais s'inspirer ou utiliser les scripts écrits précédemment. Par exemple, en exécutant cette fonction avec les angles de 2023 rad et 2024 rad, on aurait :

```
#main
angle = 2024
c = cosinus(angle)
print(f"cos({angle}) = {c}")

mainloop()
```



```
>>> (executing file "tdTrigo.py")
cos(2023)=0.98
```



```
>>> (executing file "tdTrigo.py")
cos(2024) = 0.6799999999999999
```

NB : Pour noircir une zone graphique, on exécute les fonctions `begin_fill()` et `end_fill()` avant et après le tracé d'une zone fermée.

Q9. Ecrire le script d'une fonction nommée *sinus( )* qui prend en argument un angle en radians et retourne son sinus en affichant dans une fenêtre graphique, le secteur angulaire noirci de sa mesure principale.

5- SCRIPT DES FONCTIONS ARCCOS() ET ARCSIN() ---------- BONUS ----------

Q10. Ecrire le script d'une fonction nommée arc*Cosinus( )* qui prend en argument un nombre réel x compris entre -1 et 1 pour retourner 2 angles α compris entre $-\pi$ et π pour lesquels on a $cosinus(\alpha) = x$. Bien sûr, ne pas utiliser la fonction *acos( )* disponible dans la bibliothèque *math.* Aucun tracé graphique n'est exigé.

Q11. Ecrire le script d'une fonction nommée arc*Sinus( )* qui prend en argument un nombre réel y compris entre -1 et 1 pour retourner 2 angles α compris entre $-\pi$ et π pour lesquels on a $sinus(\alpha) = x$. Bien sûr, ne pas utiliser la fonction *asin( )* disponible dans la bibliothèque *math.*