# MOOC Intro POO C++
# Exercices semaine 1

---

### Exercice 1 : cercles (niveau 1)

> Cet exercice correspond à l'exercice n°46 (pages 109 et 291)
> de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

Le but de cet exercice est d'écrire une classe représentant la notion de cercle.

Écrivez un programme `Cercle.cc` dans lequel vous définissez une classe `Cercle` ayant comme attributs un rayon et les coordonnées du centre (du cercle).

Déclarez ensuite les méthodes «get» et «set» correspondantes, par exemple :

```
void getCentre(double& x, double& y) const { ... }
void setCentre(double x, double y) { ... }
...
```

`NOTE :` ceci n'est qu'*un* exemple parmi d'autres. Si vous préférez d'autres prototypes pour ces méthodes, libre à vous d'implémenter votre solution.
Vous pourrez en particulier réviser votre programme après avoir fait les exercices 3 et 4.

Ajoutez ensuite les méthodes (faisant partie de l'interface) :

- `double surface() const` qui calcule et retourne la surface du cercle (pi fois le carré du rayon);
- `bool estInterieur(double x, double y) const` qui teste si le point de coordonnées (x,y) passé en paramètre fait ou non partie du cercle (frontière comprise : disque fermé).

La méthode retournera `true` si le test est positif, et `false` dans le cas contraire.

Dans le `main()`, instanciez deux objets de la classe `Cercle`, affectez des valeurs de votre choix à leur attributs et testez vos méthodes `surface` et `estInterieur`.

Remarque : la constante pi est souvent définie comme `M_PI` dans le module `cmath` (`#include <cmath>` au début de votre programme).
Si ce n'est pas le cas pour votre compilateur, ajoutez simplement les lignes suivantes en début de votre programme, après les `#include` et le `using namespace std;` :

```
#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif
```

## Exercice 2 : petit tour de magie (niveau 2)

Cet exercice correspond à l'exercice n°48 (pages 111 et 296)
de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

### 2.1 Organisation de l'exercice

Dans cet exercice, nous vous demandons d'élaborer un programme orienté objet de manière indépendante pour la première fois. Nous avons donc organisé l'exercice en 2 parties :

1. une première (de 2.2 et 2.3) qui décrit le problème à résoudre et qui, idéalement devrait suffire pour élaborer puis écrire le programme de façon autonome, certainement en plusieurs tentatives ;
2. une seconde (partie 2.4), pour aider au cas où la première partie vous semble insuffisante, vu que c'est votre première conception autonome d'un programme. Dans un premier temps, essayez de ne pas la lire et revenez-y si nécessaire.

### 2.2 Buts du programme

On souhaite ici écrire un programme « simulant » le tour de magie élémentaire suivant :

> *Un magicien demande à un spectateur d'écrire sur un papier son âge et la somme qu'il a en poche (moins de 100 francs).*
> *Il lui demande ensuite de montrer le papier à son assistant, qui doit le lire (sans rien dire), puis effectuer secrètement le calcul suivant : multiplier l'âge par 2, lui ajouter 5, multiplier le résultat par 50, ajouter la somme en poche, et soustraire le nombre de jours que contient une année, puis finalement donner le résultat à haute voix.*
> *En ajoutant mentalement (rapidement !) 115 au chiffre reçu, le magicien trouve tout de suite l'âge et la somme en poche (qui étaient restés secrets).*

Modéliser ce tour de magie, en définissant au moins les classes (simples) `Magicien`, `Assistant` et `Spectateur`. Il pourrait également être utile de disposer d'une classe `Papier`.

L'instance de `Spectateur` devra demander son âge à l'utilisateur du programme ainsi que la somme d'argent en poche, et s'assurer qu'une valeur correcte est entrée (entre 0 et 99).

Essayer de faire une modélisation la plus exacte possible ; faire notamment usage des droits d'accès là où cela semble pertinent. Pour chaque méthode, effectuer un affichage à l'écran de l'opération en cours et de l'acteur qui la réalise.

**Note :** Il existe de nombreuses variantes possibles. Commencer par un modèle très simple, et le faire évoluer pour se rapprocher de la situation « réelle » décrite.

### 2.3 Exemple de déroulement

```
[Spectateur] (j'entre en scène)
Quel âge ai-je ? 35
Combien d'argent ai-je en poche (<100) ? 112
Combien d'argent ai-je en poche (<100) ? 12
[Spectateur] (je suis là)
[Magicien] un petit tour de magie...
[Spectateur] (j'écris le papier)
[Spectateur] (je montre le papier)
[Assistant] (je lis le papier)
[Assistant] (je calcule mentalement)
[Assistant] J'annonce : 3397 !
[Magicien]
  - hum... je vois que vous êtes agé de 35 ans
    et que vous avez 12 francs en poche !
```

### 2.4 Indications plus détaillées

La page suivante contient quelques indications en vrac qui peuvent vous être utiles. Ne les lisez pas si vous voulez être complètement indépendant (but premier de l'exercice)...

---

- Les « objets » du programme ont déjà été suggérés : `Magicien`, `Assistant`, `Spectateur` et éventuellement `Papier` ; réfléchissez alors aux attributs : « qui *a* quoi ? » et surtout aux méthodes : « qui *fait* quoi ? »
- Essayez de définir une méthode pour chaque action élémentaire effectuée : premières actions du spectateur (demander l'âge et la somme d'argent), écrire sur le papier, montrer le papier ; pour l'assistant, lire le papier (qu'il doit donc recevoir), faire le calcul ; etc.
- Procédez par étapes, petit à petit.

---

## Exercice 3 : coordonnées 3D (niveau 1)

Cet exercice correspond à l'exercice n°47 (pages 110 et 294)
de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

Le but de cet exercice est d'implémenter de façon élémentaire une classe représentant les coordonnées dans l'espace (3D).

Dans le fichier `Point3D.cc`, définir la classe `Point3D` représentant un point dans l'espace par ses trois coordonnées, et possédant les méthodes suivantes :

- « `init` », permettant d'initialiser les trois coordonnées d'un objet `Point3D` à partir de trois valeurs de type `double` reçus en paramètres ;
- « `affiche` », permettant l'affichage de coordonnées d'un objet `Point3D` ;
- et « `compare` », permettant de comparer (tester l'égalité des coordonnées) l'objet courant à un autre objet de type `Point3D` passé en paramètre.

Dans le `main`, créer trois points (trois instances) dont deux avec des coordonnées identiques et tester les deux méthodes précédentes.

**Exemple d'utilisation (`main()`)**

```
Point3D point1;
Point3D point2;
Point3D point3;

point1.init(1.0, 2.0, -0.1);
point2.init(2.6, 3.5,  4.1);
point3 = point1;

cout << "Point 1 :";
point1.affiche();

cout << "Point 2 :";
point2.affiche();

cout << "Le point 1 est ";
if (point1.compare(point2)) {
  cout << "identique au";
} else {
  cout << "différent du";
}
cout << " point 2." << endl;

cout << "Le point 1 est ";
if (point1.compare(point3)) {
```

```
  cout << "identique au";
} else {
  cout << "différent du";
}
cout << " point 3." << endl;
```

---

## Exercice 4 : triangles (niveau 2)

### 4.1 Organisation de l'exercice

Comme dans l'exercice 2, nous vous demandons dans cet exercice d'élaborer un programme orienté objets de manière indépendante. Nous l'avons donc à nouveau organisé en deux parties :

1. une première (de 4.2 à 4.4) qui décrit le problème à résoudre et qui, idéalement devrait suffire pour élaborer puis écrire le programme de façon autonome ;
2. une seconde (partie 4.5), pour aider au cas où la première partie vous semble insuffisante. Essayez dans un premier temps de ne pas la lire et revenez-y si nécessaire.

### 4.2 Buts du programme

Sur la base du programme de l'exercice précédent, écrire un nouveau programme `triangles.cc` qui permet à l'utilisateur d'entrer les coordonnées ($x$, $y$ et $z$) des sommets d'un triangle (en 3D). Le programme affiche ensuite le périmètre du triangle ainsi qu'un message indiquant s'il s'agit ou non d'un triangle isocèle.

### 4.2 Indications générales (aide mathématique)

- Un triangle est isocèle si au moins deux côtés ont la même longueur.
- La formule pour calculer la distance entre deux points de l'espace $(x_1, y_1, z_1)$ et $(x_2, y_2, z_2)$ est : la racine carrée (fonction `sqrt` de `cmath`) de $( (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2 )$
- Le périmètre d'un triangle peut être calculé comme la somme des distances entre les trois sommets.

### 4.4 Exemple de déroulement

```
Construction d'un nouveau point
   Veuillez entrer x : 0
   Veuillez entrer y : 0
   Veuillez entrer z : 0
Construction d'un nouveau point
   Veuillez entrer x : 2.5
   Veuillez entrer y : 2.5
   Veuillez entrer z : 0
Construction d'un nouveau point
   Veuillez entrer x : 0
   Veuillez entrer y : 5
   Veuillez entrer z : 0
Périmètre : 12.071067811865476
Ce triangle est isocèle !
```

### 4.5 Indications plus détaillées

La page suivante contient quelques indications en vrac qui peuvent vous être utiles. Ne les lisez pas si vous voulez être complètement indépendant (but premier de l'exercice)...

---

- Réfléchissez aux objets que vous devez utiliser dans le programme. Vous pourriez par exemple représenter le triangle par une classe `Triangle` qui contient les longueurs de ses trois cotés, puisqu'au final nous n'utilisons pas les points du triangle en tant que tels mais simplement les longueurs de ses cotés.
- Réfléchissez également aux méthodes qui seraient utiles pour les classes `Triangle` et `Point3D` (laquelle ne servirait que temporairement pour initialiser un triangle).
- Définissez une fonction `distance` qui prend deux `Point3D` en arguments et retourne leur distance.

---