GCC ! – Prologin

Avant de commencer

— Si tu ne comprends pas quelque chose, n'hésite surtout pas à demander à un orga ! Nous sommes là pour ça !
— On te conseille de créer un fichier par exercice, tous dans le même dossier.
— On te conseille fortement de tester tous les codes présents dans ce TP, même s'ils ne sont que des exemples. Si tu ne comprends pas comment ils fonctionnent, ou que quelque chose cloche, n'hésite pas à appeler un orga.

La commande print

D'habitude, le tout premier programme qu'on écrit quand on commence à apprendre un langage de programmation sert à afficher "Hello world !" à l'écran. On va commencer par là nous aussi. Ouvre un nouveau fichier et enregistre-le sous le nom `hello_world.py` > On évite de mettre des espaces dans les noms de fichier. On les remplace par le symbole underscore `_`. On évite aussi les majuscules.

> Le ".py" s'appelle une extension. Cela sert à préciser à l'ordinateur que ce qui est écrit dans le fichier est écrit dans le langage python, ce qui peut permettre à certains éditeurs de colorer les mots clefs de Python.

Dans ce fichier, écris

```
1 print("Hello world !")
```

et enregistre.

Maintenant, nous allons exécuter ce programme. Pour cela, il suffit de cliquer sur "Run" et tu verras l'écran se diviser en deux. Sur la partie basse, s'affiche le résultat de ton programme.

Comme on le voulait, la phrase Hello world ! s'affiche. Maintenant, l'exécution de notre programme est terminée.

Découverte des erreurs

Pour donner des instructions à l'ordinateur en utilisant Python, on doit utiliser un langage précis et codifié. Observons notre première instruction : le message qu'on veut écrire doit être entouré de guillemets et être entre parenthèses.

Essaie d'enlever les parenthèses ou les guillemets et d'exécuter à nouveau le programme (n'oublie pas d'enregistrer quand tu as fini de modifier !).

Comme tu peux le constater, au lieu d'afficher Hello world ! Python renvoie ce qu'on appelle une erreur : c'est un message pour expliquer que le programme a un problème. Le plus souvent, il précise la nature de l'erreur et la ligne où il l'a détectée. Mais attention, ce n'est pas forcément à cette ligne qu'on devra modifier quelque chose pour que le programme fonctionne.

Exercice :

Écris un programme qui affiche la phrase :

```
Bienvenue au stage d'informatique GCC! !
```

Exécute-le pour vérifier qu'il fonctionne.

Variables

Nous allons maintenant demander à Python de retenir des valeurs pour nous et de faire des opérations avec ces valeurs. Pour cela, on va utiliser ce qui s'appelle une variable.

Quand on utilise Python, il faut s'imaginer qu'on a accès à un grand meuble avec plein de tiroirs. Ces tiroirs sont les variables. Sur chaque tiroir il y a une étiquette : c'est le nom de la variable.

Dans un tiroir, on peut mettre une valeur. On peut aussi ouvrir le tiroir pour lire la valeur qu'on a mise à l'intérieur. Ou encore, on peut ouvrir le tiroir et remplacer la valeur qu'on y a mise par une autre.

Pour créer une variable avec Python, on écrit par exemple :

```
1 x = 2
```

Cela veut dire "ouvre le tiroir avec l'étiquette x et range la valeur 2 à l'intérieur".

Crée un nouveau fichier `variable.py` et entre cette ligne de code, puis exécute-le.

Il ne se passe rien ! C'est normal, on n'a pas demandé à Python d'afficher quoi que ce soit. Rajoute maintenant `print(x)`

et exécute à nouveau le programme. Tu vois que Python affiche la valeur contenue dans la variable x, ici 2.

Opérations

Python nous permet d'effectuer des opérations, comme une calculatrice très perfectionnée. Essaie le programme suivant :

```
1 print(4+3)
```

Tu remarques qu'ici, il n'y a pas de guillemets, comme avant lorsqu'on a écrit print(x). C'est parce qu'on ne met des guillemets que lorsqu'on veut afficher une chaîne de caractères. Tu peux essayer de voir ce qui se passe si tu écris plutôt `print(«x»)` ou `print(«4+3»)`.

Mais ce qui est encore plus pratique, c'est qu'on peut faire des opérations avec les valeurs contenues dans nos variables. Essaie par exemple le programme suivant :

```
1 a = 4
2 b = 3
3 resultat = a+b
4 print(resultat)
```

À quoi ça sert ? Ici, le calcul est très simple, mais imagine que tu aies beaucoup d'opérations à effectuer. En écrivant ton programme avec des variables, tu n'as qu'à changer leur valeur au début pour pouvoir refaire toutes ces opérations sans avoir à écrire de nouveau du code. Si tu voulais plutôt calculer 5+3 ici, il suffirait de remplacer la première ligne par a=5. Cela va devenir encore plus clair avec le prochain paragraphe, où tu verras qu'on peut demander à l'utilisateur un nombre de son choix qu'on mettra ensuite dans une variable.

Il s'agit de ton premier programme avec plusieurs instructions, et donc plusieurs lignes, car en python, on met une instruction par ligne. Une instruction, c'est une tâche qu'on demande à l'ordinateur d'effectuer : affiche ceci, mets cette valeur dans cette variable, etc.

Mais Python ne permet pas seulement de faire des additions. Tu peux aussi utiliser les signes `-`, `*`, `/` pour effectuer des soustractions, des multiplications et des divisions. N'hésite pas à essayer !

Pour aller plus loin :

En plus des quatre opérations classiques, python propose deux autres opérateurs : `//` et `%`.

Essaie-les pour comprendre ce qu'ils font.

Exercice :

Voici un programme python :

```
a = 7
b = 3
c = a*2
d = c+b
print(d-4)
```

Que va afficher l'ordinateur à la fin de ce programme ? Vérifie en l'essayant.

Exercice :

Voici un programme python :

```
x = 3
x = x - 2
print(x)
```

Que va afficher l'ordinateur à la fin de ce programme ? Vérifie en l'essayant.

Exercice :

On cherche à écrire un programme python qui inverse le contenu de deux variables. On propose le programme suivant :

```
a = 4
b = 6
a = b
b = a
print("a vaut" + str(a))
print("b vaut" + str(b))
```

1. À ton avis, que va afficher l'ordinateur ? Essaie pour vérifier.
2. Écris un programme qui permet d'inverser le contenu de deux variables.

   Pour afficher une chaîne de caractères (du texte, entre guillemets) et un nombre dans la même phrase, on va devoir dire à l'ordinateur de traiter le nombre comme du texte. C'est ce que veut dire str(…) Le signe + entre deux morceaux de texte veut dire qu'on va les mettre l'une à la suite de l'autre. Cette opération s'appelle concaténer.

Demander une valeur à l'utilisateur

Pour demander une valeur à l'utilisateur de notre programme, on va utiliser l'instruction `input`. Cette instruction dit à l'ordinateur de suspendre le programme jusqu'à ce que l'utilisateur entre une réponse et appuie sur entrée.

Nous allons essayer le programme suivant :

```
1 print("Comment t'appelles-tu ?")
2 nom = input()
3 print("Tu t'appelles" + nom)
```

Maintenant, nous allons demander à l'utilisateur d'entrer une valeur chiffrée. Par exemple, demandons-lui son âge. Pour cela, plutôt que d'écrire `nom = input()`, on va écrire `age = int(input())`. Quand on demande un nombre à l'utilisateur, il faut utiliser `int(input())` et pas seulement `input()`.

Pour aller plus loin : Pourquoi ce `int()` ?

python ne traite pas de la même manière les nombres entiers, les nombres à virgule, et le texte. Quand `input()` récupère l'entrée de l'utilisateur, il la traite comme une chaîne de texte. Or, on ne peut pas faire certaines opérations sur les chaînes de texte (soustraction, division, etc.) dont on aurait besoin. Le `int()` dit à l'ordinateur qu'en fait, on veut traiter ce que l'utilisateur entre comme un nombre. La chaîne de texte "4" devient ainsi le nombre 4. C'est le même mécanisme dans l'autre sens qu'on a vu avec `str()` avant.

Le programme sera alors :

```
1 print("Quel est ton âge ?")
2 age = int(input())
3 print("Tu as " + str(age) + " ans.")
```

N'oublie pas qu'une fois la valeur entrée dans une variable, on peut faire des opérations avec elle, et pas seulement l'afficher.

> Nous allons maintenant faire quelques exercices pour manipuler les variables ainsi que les instructions print et input. Si tu as l'impression que les petits programmes que nous écrivons pour l'instant ne servent pas à grand-chose, ne t'inquiète pas : c'est normal ! Pour l'instant, on essaie simplement de bien comprendre comment tout fonctionne, mais tu verras qu'à la fin du stage, ou sur les exercices bonus de ce TP, tu seras capable d'écrire des programmes qui font des tâches très complexes et qui te simplifieront la vie ou te permettront de faire des choses impossibles sans l'informatique.

Exercice :

Écris un programme qui pose trois questions à l'utilisateur : son jour de naissance, son mois de naissance et son année de naissance. On devra attendre que l'utilisateur ait répondu pour passer à la question suivante. Quand on aura les trois réponses, le programme doit alors afficher : `Vous êtes né le jour mois année`

Exercice :

Écris un programme qui demande son âge à un utilisateur et qui affiche ensuite : `Vous aurez 18 ans dans … ar`

> Et si l'utilisateur rentre un nombre supérieur à 18 ? Essaie ! Ce n'est sans doute pas comme ça qu'on voudrait que le programme se comporte. On aimerait plutôt qu'il affiche "vous avez déjà 18 ans" ou quelque chose comme ça. C'est justement ce qu'on va apprendre à faire tout de suite !

Les conditionnelles (if – else – elif)

Jusqu'à maintenant, nos programmes se sont toujours déroulés en exécutant ligne par ligne ce qu'on leur demandait. D'abord, ils faisaient l'instruction donnée à la première ligne, puis celle de la seconde, etc. jusqu'à la dernière ligne.

Mais parfois, on voudrait pouvoir écrire un morceau de programme qui n'est exécuté que si une certaine condition est remplie, et un autre si elle n'est pas remplie. Par exemple, pour reprendre l'exercice précédent, on voudrait faire quelque chose de différent selon que l'utilisateur a déjà 18 ans, ou pas encore.

Pour cela, on va utiliser ce qu'on appelle une conditionnelle. En voici un exemple :

```
1 print("Quel est ton âge ?")
2 age = int(input())
3 if age < 18 :
4     resultat = 18-age
5     print("Vous aurez 18 ans dans " + resultat + " ans.")
6 else :
7     print("Vous avez déjà 18 ans.")
```

`if` est un mot anglais qui veut dire "si" et `else` "sinon". Ce programme dit donc à l'ordinateur :

— Si l'âge est inférieur à 18 ans : calcule le nombre d'années avant d'avoir 18 ans et mets-le dans la variable résultat, puis affiche "Vous aurez 18 ans dans `resultat` ans."
— Sinon, affiche "Vous avez déjà 18 ans."

> Tu remarques que ce code a un aspect bien particulier. Il y a des `:` à la fin des lignes avec un if ou un else, et les lignes après sont indentées (on utilise la touche tab pour cela). Il faut respecter cela pour que le programme fonctionne.

Pour écrire des conditionnelles, nous allons avoir besoin de savoir écrire des conditions. On les écrit globalement comme en mathématiques :

— Pour dire “a est inférieur à b” on écrit `a < b`
— Pour dire “a est supérieur à b” on écrit `a > b`
— Pour dire “a est inférieur ou égal à b” on écrit `a <= b`
— Pour dire “a est supérieur ou égal à b” on écrit `a >= b`
— Pour dire “a est égal à b” on écrit `a == b`
— Pour dire “a est différent de b” on écrit `a!=b`

Exercice :

Imaginons que nous avons un jeu auquel on gagne une fois qu’on a un score de 21 ou plus. Écris un programme qui demande son score à l’utilisatrice puis affiche selon le cas “Tu as gagné” ou “Tu as perdu”.

Exercice :

Cet exercice va nous permettre de mieux comprendre à quoi servent les tabulations.

Pour chaque question, essaie de trouver ce que le programme python donné va afficher si on entre les nombres donnés, puis écris-le et exécute-le. Fais attention, quand tu le recopies, à bien respecter la même indentation que dans l’énoncé.

1.

```
print("Quel est ton score ?")
score = int(input())
if score   100:
    print("Bravo, tu as gagné !")
else :
    print("Tu as perdu.")
print("Merci d’avoir joué !")
```

Que se passe-t-il si on entre 50 ? Et si on entre 120 ?

2.

```python
print("À combien de stages GCC! as-tu participé ?")
nombre_stages = int(input)
if nombre_stages == 0:
    print("C'est bizarre pourtant, puisque tu es là !")
else:
    if nombre_stages == 1:
        print("Bienvenue !")
    else:
        print("Bon retour parmi nous !")
    print("Profite bien du stage !")
```

Que se passe-t-il si on entre 0 ? Et si on entre 1 ? Et si on entre 2 ?

Que faire si l'on a plus de deux cas ? Comme tu l'as vu plus haut, on peut imbriquer les structures if else. Ce sera aussi le cas pour toutes les autres structures que nous verrons et c'est très souvent utilisé.

Mais dans le cas des conditionnelles, on a un autre moyen plus facile et plus lisible. Voici la syntaxe :

```python
if condition1:
    instructions du cas 1
elif condition2:
    instructions du cas 2
elif condition3:
    instructions du cas 3
else:
    instructions dans le reste des cas
```

On peut mettre autant de elif que l'on veut. elif est la contraction de else if, sinon si

Exercice :

Réécris le programme précédent avec des `elif`.

Exercice :

Connais-tu le concours Prologin ? C'est un concours de programmation auquel on peut participer jusqu'à l'année de ses 20 ans. On voudrait écrire un programme qui demande à l'utilisateur ou l'utilisatrice son année de naissance, et dit en fonction :

— "Tu peux participer au concours Prologin."
— "C'est la dernière année où tu peux participer au concours Prologin."
— "Tu ne peux plus participer au concours Prologin." Écris-le en utilisant la structure elif. Pour cette année, les personnes nées en 2001 ou après peuvent participer.

Parfois, nous aurons besoin de combiner des conditions. Par exemple, nous voudrons écrire “condition1 ou condition2”.

Pour ce faire, en python, on utilise les mots-clefs suivants :

— `or` pour OU
— `and` pour ET
— `not` pour NON. Cela permet d’exprimer l’inverse d’une proposition. Par exemple, `not x>0` équivaut à `x <= 0`.

Exercice :

Traduis les phrases suivantes avec des conditions en python.

1. n est positif ou n = 5
2. a différent de 7 et a différent de 9
3. x est strictement compris entre 12 et 19

La boucle for

Nous allons voir ici un moyen de demander à notre ordinateur de répéter plusieurs fois un même bloc d’instructions. À nouveau, notre programme ne va pas exécuter les instructions une par une. Il y a un bloc de lignes qu’il va exécuter dans l’ordre, puis exécuter à nouveau à partir de la première, un certain nombre de fois. C’est pour ça qu’on parle de boucle.

Exercice :

```
1 for i in range(5) :
2     print("Donnez le nom d'un joueur ou d'une joueuse")
3     nom = input()
4     print("Nous avons un joueur ou une joueuse du nom de " + nom)
```

1. Que fait ce programme ?

   Comme tu peux le constater, on retrouve la construction de la conditionnelle avec les : en fin de ligne et les tabulations. Ce qu’il faut retenir de la boucle for, c’est que for i in range(n) permet d’exécuter un morceau de code n fois. On appelle une exécution du bloc de code indenté sous le for un tour de boucle. Ici, la première fois qu’on demande le nom, c’est le premier tour de boucle, la deuxième fois, c’est le deuxième tour de boucle, etc.

2. Ce programme fonctionne si on a exatement 5 joueurs ou joueuses. Mais si le nombre de joueurs ou joueuses change à chaque partie ? On ne va pas réécrire un programme à chaque fois !

Écris un programme qui demande le nombre de joueurs ou joueuses à l'utilisateur puis qui exécute la boucle un nombre adapté de fois. Indice : Rappelle-toi que dans le range, on indique un nombre, mais qu'un nombre peut être stocké dans une variable…

Tu remarques sans doute qu'on préférerait demander "Quel est le nom de la joueuse 1 ?" puis "Quel est le nom de la joueuse 2 ?" etc. Ce serait plus clair.

Eh bien, savoir à quel tour de boucle on se trouve, c'est quelque chose qui sera souvent utile. C'est pour cette raison qu'on utilise un i dans `for i in range(…)` . i est une variable comme n'importe quelle autre, qu'on peut manipuler, afficher, etc.

Attention : c'est généralement une mauvaise idée d'écrire `i =` dans le corps de la boucle. i change naturellement de valeur à chaque tour de boucle (il augmente de 1). Changer la valeur de i n'est généralement pas la bonne solution à un problème.

Note : Nous avons tout écrit avec un i, mais en fait, on peut remplacer i par n'importe quel nom de variable. Essaie avec j, ou rang, par exemple. On appelle cette variable la variable de boucle.

Exercice :

On veut modifier le programme de l'exercice précédent pour qu'il demande et affiche le numéro de la joueuse. On propose le programme suivant :

```
1 print("Combien y a-t-il de joueurs ou joueuses ?")
2 nombre_joueurs = int(input())
3 for i in range(nombre_joueurs):
4         print("Donnez le nom du joueur ou de la joueuse " + str(i))
5         nom = input()
6         print("Le joueur ou la joueuse " + str(i) + " s'appelle" + nom)
```

1. Essaie ce programme. Est-ce qu'il donne le résultat escompté ?

   On voit qu'avec range(n), la variable de boucle (i ici) varie entre 0 et n-1. La boucle est donc bien exécutée n fois, mais i ne varie pas entre 1 et n comme on pourrait s'y attendre. C'est une source courrante d'erreurs.

2. Modifie le programme pour qu'il ait le comportement attendu (on veut que les joueurs ou joueuses soient numérotées de 1 à nombre_joueurs et non de 0 à nombre_joueurs-1).

Exercice :

Écris un programme qui affiche un compteur de 1 à n, où n est un nombre entré par l'utilisateur ou l'utilisatrice.

Exercice :

Écris un programme qui affiche un compteur de 10 à 0, et termine en affichant "C'est terminé !"

Exercice :

Deux joueuses jouent au ping-pong. Écris un programme qui leur demande combien de parties elles vont faire puis, pour chaque partie, leur demande leurs scores et dit qui a gagné.

Indice : on peut mettre une structure if else dans une boucle for.

Exercice :

Maintenant, ces deux joueuses jouent au tennis, et elles voudraient afficher un message qui dit à quel set et à quel match elles sont. Par exemple, s'il y a trois sets de deux matchs chacun, notre programme affichera :

```
Nous sommes au set 1, match 1
Nous sommes au set 1, match 2
Nous sommes au set 2, match 1
Nous sommes au set 2, match 2
Nous sommes au set 3, match 1
Nous sommes au set 3, match 2
```

Écris un programme qui :

— demande à l'utilisateur ou utilisatrice combien il y aura de sets
— demande à l'utilisateur ou utilisatrice combien il y aura de matchs par set
— affiche les phrases sur le modèle ci-dessus.

Pour aller plus loin :

En fait, `range` peut prendre plus d'un paramètre. On peut utiliser `for i in range(a,b)` pour faire varier `i` entre a et b-1. On peut même utiliser `for i in range(a,b,p)` pour faire varier `i` entre a et b-1 avec un pas de p. Par exemple, pour un compte à rebours, on peut prendre `p = -1` ou pour compter de deux en deux, `p = 2`.

La boucle while

La boucle while est une structure qui ressemble à la boucle for. Comme avant, on va exécuter des instructions en boucle, d'où le nom de la structure.

Cependant, dans la boucle for, on exécutait la boucle un nombre précis de fois. Ce nombre pouvait être entré par l'utilisateur ou l'utilisatrice, mais une fois qu'on commençait la boucle, il ne bougeait plus.

Or, parfois, on a envie d'exécuter une boucle un nombre de fois qu'on ne peut pas déterminer à l'avance. C'est à ça que sert la boucle while. Elle effectue le même bloc d'instructions tant que (while en anglais) une certaine condition est remplie.

Par exemple, on peut faire un programme qui envoie un message d'encouragement tant que l'utilisateur ou l'utilisatrice le demande :

```
print("As-tu besoin d'encouragements ? Si oui, tape 1, si non, tape 2.")
reponse = int(input())
while reponse == 1 :
    print("Tu vas y arriver ! J'ai confiance en toi !")
    print("As-tu besoin d'encouragements ? Si oui, tape 1, si non, tape
    ↪  2.")
    reponse = int(input())
```

La syntaxe de la boucle while est la même que celles que nous avons vues précédemment : le `:` , des indentations. Retiens aussi qu'on peut utiliser les structures précédentes (la boucle for et les conditionnelles) avec la boucle while.

Exercice :

On joue à un jeu de hasard. L'utilisateur ou utilisatrice doit deviner un nombre entre 1 et 10. Tant qu'il ou elle n'a pas trouvé, on lui redemande de proposer un nombre, et on s'arrête quand il ou elle trouve le nombre secret. Écris un programme permettant de jouer à ce jeu.

Exercice :

Maintenant, on veut que le joueur ou la joueuse devine un nombre secret entre 1 et 100. Mais comme c'est un peu compliqué, on va lui dire à chaque essai, s'il a proposé un nombre trop grand ou trop petit. Un exemple d'échange serait le suivant :

```
Proposez un nombre
40
Trop petit !
50
Trop petit !
83
Trop grand !
74
Bravo, vous avez deviné !
```

Écris un programme qui permet de jouer à ce jeu.

Bonus : À ton avis, quelle est la meilleure stratégie pour gagner à ce jeu ?

Exercice :

Boucle for ou boucle while

On veut écrire un programme qui affiche les entiers pairs entre 0 et 2n, où n est un nombre donné par l'utilisateur ou l'utilisatrice.

1. Écris un tel programme en utilisant une boucle for.
2. Écris un tel programme en utilisant une boucle while.

Les fonctions

Si tu as déjà vu les fonctions en cours de mathématiques, cette partie devrait te rappeler quelque chose. Sinon, voilà une petite explication de ce que sont les fonctions en python.

Une fonction est comme une boîte noire. On lui donne un ou plusieurs objets qu'on appelle les entrées, et elle nous donne en retour un ou plusieurs objets qu'on appelle sorties et/ou elles font une action (afficher un message par exemple). Et à l'intérieur, eh bien, on ne sait pas forcément ce qui se passe. Bien sûr, quand nous écrirons nos propres fonctions, on saura ce qu'elles font ! Mais tu as déjà utilisé des fonctions sans le savoir : `print` et `input`. Ces fonctions ne renvoient rien, elles n'ont pas de sortie. Cela arrive.

`print` prend en entrée un nombre, ou du texte, et affiche cette entrée à l'écran.

`input` ne prend pas d'entrée, et ne renvoie rien, mais permet de récupérer une valeur entrée par l'utilisatrice.

On sait ce que font ces deux fonctions, mais pas comment elles le font. C'est ce qu'on veut dire par boîte noire. Beaucoup de lignes de code se cachent derrière `print` et `input` mais on n'a pas besoin de les connaître. Savoir quel effet ont ces deux fonctions suffit à les utiliser.

Nous allons écrire une fonction qui prend en entrée un nombre `x` et renvoie son double.

```
1 def double(x):
2     resultat = 2*x
3     return resultat
```

Discutons maintenant de la forme de cette fonction :

— On retrouve les `:` et les tabulations comme dans toutes les autres structures
— Les entrées, ici `x`, sont indiquées entre parenthèses. Si on veut mettre plusieurs entrées, il faut les séparer avec des virgules, par exemple : `f(x,y)`.
— Le mot-clef `return` sert à indiquer la sortie de la fonction. Quand on atteint une ligne avec return, on sort du code de la fonction, le reste n'est pas exécuté.

Maintenant, si on tape `print(double(4))` on obtient :

```
8
```

Pour donner une valeur en entrée à une fonction, ici `4`, on doit aussi le mettre entre parenthèses, après le nom de la fonction.

Exercice :

Voici une fonction.

```
def f(x):
    resultat = x + 3
    return resultat
```

Quel est la valeur de `f(17)` ? De `f(0)` ? Vérifie en écrivant la fonction sur ton ordinateur.

Exercice :

Voici une fonction.

```
def f(x):
    if x <= 5:
        resultat = x * x
    else :
        resultat = x*5
    return resultat
```

Quel est la valeur de `f(3)` ? De `f(5)` ? De `f(7)` ? Vérifie en écrivant la fonction sur ton ordinateur.

Exercice :

Écris une fonction qui prend en entrée un nombre et renvoie le triple de ce nombre.

> À quoi sert une fonction ? Pas seulement à faire des calculs ! Avoir des fonctions, c'est pouvoir diviser son programme en petits morceaux autonomes. Pour programmer avec moins d'erreurs, on va souvent découper notre programme en fonctions. Nous allons voir des exemples dans le prochain exercice, en reprenant un de nos programmes précédents pour l'écrire avec des fonctions.

Exercice :

Reprenons le jeu où on doit deviner un nombre secret entre 1 et 100.

1. Écris une fonction `tour(secret, nombre)` qui affiche selon le cas “Trop petit !”, “Trop grand !” ou “Bravo, tu as gagné !”

Indice : cette fonction ne contiendra pas de `return`.

2. Écris maintenant un programme qui permet de jouer au jeu en utilisant ta fonction.

Problème : Le jeu de Nim

Nous allons coder un programme qui permet à deux joueurs ou joueuses de s'affronter au jeu de Nim. Les règles sont très simples : on commence avec vingt bâtons, et chacun son tour, on en retire un, deux, ou trois de la pile. La personne qui prend le dernier bâton gagne. Nous voudrions que le résultat du programme ressemble à ça :

```
Il y a 20 bâtons.
Joueur ou joueuse 1, c'est à toi ! Combien de bâtons veux-tu enlever ?
3
Il y a 17 bâtons.
Joueur ou joueuse 2, c'est à toi ! Combien de bâtons veux-tu enlever ?
2
Il y a 15 bâtons.
Joueur ou joueuse 1, c'est à toi ! Combien de bâtons veux-tu enlever ?
3
Il y a 12 bâtons.
Joueur ou joueuse 2, c'est à toi ! Combien de bâtons veux-tu enlever ?
2
Il y a 10 bâtons.
Joueur ou joueuse 1, c'est à toi ! Combien de bâtons veux-tu enlever ?
1
Il y a 9 bâtons.
Joueur ou joueuse 2, c'est à toi ! Combien de bâtons veux-tu enlever ?
1
Il y a 8 bâtons.
Joueur ou joueuse 1, c'est à toi ! Combien de bâtons veux-tu enlever ?
3
Il y a 5 bâtons.
Joueur ou joueuse 2, c'est à toi ! Combien de bâtons veux-tu enlever ?
3
Il y a 2 bâtons.
Joueur ou joueuse 1, c'est à toi ! Combien de bâtons veux-tu enlever ?
2
Il ne reste plus de bâtons. Joueur ou joueuse 1, tu remportes la partie !
```

1. Crée une fonction `tour(numero_joueuse,nb_batons)` qui permet d'afficher :

```
Il y a nb_batons batons.
Joueur ou joueuse numero_joueuse, c'est à toi ! Combien de bâtons veux-tu
↪  enlever ?
```

et qui renvoie le nouveau nombre de bâtons.

2. Avec cette fonction, écris un programme permettant de jouer au jeu de Nim.

Indice : Il y a un nouveau tour tant que le nombre de bâtons n'est pas égal à 0.

3. Crée une fonction `jeu_de_Nim` avec ton précédent programme qui permet de jouer à une partie du jeu. Ensuite, crée un programme qui permet de jouer tant que le joueur ou la joueuse le désire.

Bonus : Il y a une manière de gagner à coup sûr à ce jeu si on est le premier joueur ou la première joueuse. Essaie de la trouver.