Tous les programmes doivent être testés : lorsqu'ils dépendent de la valeur d'une ou plusieurs variables (exercices 4 et suivants), on testera le programme pour des valeurs bien choisies de ces variables.
Les programmes seront écrits dans l'éditeur.

I) Quelques types

A) Chaînes de caractères

Exercice 1
Écrire un programme qui demande à l'utilisateur de rentrer une chaîne de caractères, que l'on affectera dans une variable.

1) Puis ce programme affiche (`print`) le contenu de la variable ;

2) Puis il affiche une phrase d'explication sur la même ligne que la valeur de la variable (par exemple « Voici le contenu de la variable [nom que vous avez choisi] »).

3) Puis, en supposant que l'utilisateur ait entré un entier (on lui demandera par une phrase ad hoc, qui se présentera sous forme de chaîne de caractères entre les parenthèses du `input()`), convertir la chaîne de caractères en entier.

B) Booléens

Exercice 2
À l'aide du TP précédent, donner la liste des opérations que vous connaissez sur les nombres ayant pour résultat un booléen.

Exercice 3
Que rend la commande `(.1+.1+.1)*70==21` ? Expliquer, puis proposer un test plus pertinent.

Morale : Tester une égalité sur des flottants, qui sont intrinsèquement des approximations, n'a pas beaucoup de sens, en plus de conduire quasi certainement à des catastrophes. □

Exercice 4
Écrire les expressions booléennes traduisant les conditions suivantes. Les variables sont toutes de type flottant. Dans un premier temps il faut formaliser le problème : obtenir une formule sur le papier. Dans un deuxième temps on traduit en Python. On testera ses expressions.

1) Le point de coordonnées (x, y) est à l'intérieur du cercle de centre (xc, yc) et de rayon r.
Pour obtenir une racine, il est conseillé d'utiliser `**(1/2)`. Les plus aventureux pourront essayer de comprendre ce que signifie `from math import sqrt`.

2) Les points de coordonnées $(x1, y1)$ et $(x2, y2)$ sont situés sur une même droite parallèle à l'un des axes du repère.

3) Les points de coordonnées $(x1, y1)$ et $(x2, y2)$ sont les sommets opposés d'un carré dont les côtés sont parallèles aux axes du repère.

Mêmes questions, mais avec des variables de type entier.

1) L'entier n est divisible par 5.

2) Les entiers n et m sont multiples l'un de l'autre.

3) Les entiers n et m sont de même signe. On proposera deux solutions. Laquelle vous semble la plus rapide pour la machine ?

II) Branchements

1) Rappels de syntaxe

Pour séparer des blocs de commandes, le texte est indenté, c'est-à-dire décalé vers la droite. Voici un exemple de texte indenté. Il y a trois niveau d'indentation. Le texte « normal » est au niveau 0, le texte en italique au niveau 1, et le texte en petit au niveau 2.

> Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus.
>
> > *Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi.*
> >
> > *Proin porttitor, orci nec nonummy molestie, enim est eleifend mi, non fermentum diam nisl sit amet erat.*
> >
> > > Cras vestibulum bibendum augue.
> > > Praesent egestas leo in pede. Praesent blandit odio eu enim.
> > > Pellentesque sed dui ut augue blandit sodales.
> > >
> > > Vestibulum ante ipsum primis in faucibus orci luctus.
> >
> > *Mauris ac mauris sed pede pellentesque fermentum. Maecenas adipiscing ante non diam sodales hendrerit. Pst canata tari est paroy malardi.*
>
> Nulla et sapien. Integer tortor tellus, aliquam faucibus, convallis id, congue eu, quam.
> Mauris ullamcorper felis vitae erat.
> Proin feugiat, augue non elementum posuere.

On aurait aussi bien pu l'écrire ainsi, ça ne change rien :

> Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus.
>
> > *Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi.*
> >
> > *Proin porttitor, orci nec nonummy molestie, enim est eleifend mi, non fermentum diam nisl sit amet erat.*
> >
> > > Cras vestibulum bibendum augue.
> > > Praesent egestas leo in pede. Praesent blandit odio eu enim.
> > > Pellentesque sed dui ut augue blandit sodales.
> > >
> > > Vestibulum ante ipsum primis in faucibus orci luctus.
> >
> > *Mauris ac mauris sed pede pellentesque fermentum. Maecenas adipiscing ante non diam sodales hendrerit. Pst canata tari est paroy malardi.*
>
> Nulla et sapien. Integer tortor tellus, aliquam faucibus, convallis id, congue eu, quam.
> Mauris ullamcorper felis vitae erat.
> Proin feugiat, augue non elementum posuere.

En Python, l'indentation a du sens, elle délimite des blocs d'instructions.

```
if < condition 1 > :
      < instruction conditionnelle 1 >
      < instruction conditionnelle 2 >
      < instruction conditionnelle 3 >
< instruction certaine 1 >
< instruction certaine 2 >
```

Les instructions que j'ai appelée conditionnelles sont considérées comme étant le bloc d'instruction du « alors » *parce qu'elles sont indentés*. On dit que l'indentation a du sens, qu'elle est *signifiante*.
On peut avoir plusieurs niveaux si on imbrique les `if` :

```
if < condition 1 > :
    < instruction conditionnelle 1 >
    if < condition 2 > :
        < instruction1.c >
        < instruction2.c >
    < instruction conditionnelle 3 >
else :
    < instruction conditionnelle du else 1 >
< instruction certaine 1 >
< instruction certaine 2 >
```

Le bloc « `if < condition 2 > :` » jusqu'à la fin de l'indentation de niveau deux joue le rôle d'une `< instruction conditionnelle 2 >`.

Pour trouver le maximum entre deux nombres, on écrirait :

```
if x < y :
  max = y
else :
  max = x
print(max)
```

Un raffinement du Si/alors/sinon : `elif`.

```
if <condition1> :
  [bloc d'instructions1]
elif <condition2> :
  [bloc d'instructions2]
else :
  [bloc d'instructions3]
```

Le déroulement de la boucle est le suivant :

1) La `<condition1>` est calculée. Si elle est vraie, le `[bloc d'instructions1]` est exécuté et la suite ignorée.

2) Si la `<condition1>` est fausse et la `<condition2>` est vraie, seules le `[bloc d'instructions2]` est exécuté et la suite ignorée.

3) Si la `<condition1>` et la `<condition2>` sont fausses, Python exécute le `[bloc d'instructions3]`

2) Exercices

Exercice 5

1) Écrire un programme qui teste si un nombre, entré par l'utilisateur (se reporter à l'exercice 1), est divisible par 2. Le résultat sera transmis par le programme via l'affichage d'une jolie phrase.

2) Écrire un programme qui teste si un nombre entré par l'utilisateur est divisible par 2 ou 3.

3) Analyser le problème et donner une réponse plus élégante et plus complète.

Exercice 6

Écrire un programme qui affecte dans la variable `maximum` (toujours donner des noms explicites !) le maximum de trois nombres x, y et z.

Morale : Bien programmer, c'est écrire des programmes qui
- fonctionnent
- font ce que l'on veut (et pas autre chose)
- sont efficace (on le reverra lorsqu'on parlera de complexité)
- sont *lisible* : noms de variables cohérents et explicites, commentaires, choix de rédaction.

□

Exercice 7 (limitations de vitesse)

Dans ce programme, l'utilisateur rentre le type de voie sur lequel il circule (agglomération, route ou autoroute — ceux qui veulent peuvent raffiner), puis il rentre sa vitesse.
Le programme affiche à la fin le contenu d'une variable `exces` qui vaut `True` ou `False`.

1) Faire un programme qui fonctionne et rempli les objectifs.

2) Améliorer le programme en supprimant les redondances et en le rendant plus lisible.

III) Boucles conditionnelles

Si vous avez terminé tout le reste. Nous verrons de toute façon tous les boucles au prochain TP.
Voici un exemple de syntaxe d'une boucle `while`.

```
i=10
S=0
while (i >= 0):
  S=S+i
  i=i-1
print(i)
```

Exercice 8
Que fait la boucle précédente ? Programmer une boucle qui affiche l'écriture binaire d'un nombre n.