Langage et programmation
2 - Constructions élémentaires

Sources entre autres : https://pixees.fr/informatiquelycee/n_site/nsi_prem_pythonbase.html

1- Prise en main de Pyzo

À faire vous-même 1

Dans la fenêtre "éditeur", saisissez le programme suivante :

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

Cliquez sur " Exécuter ", " Démarrer le script " afin d'exécuter le programme qui vient d'être saisi.

Pyzo va vous demander d'enregistrer le programme, enregistrez-le dans un dossier qui vous servira de dossier de travail

Vous devez voir le message "Hello world !!" apparaître dans la console, le shell.

2 - Notion de variable

Définition du mot ordinateur d'après "Le Petit Larousse" :

"Machine automatique de traitement de l'information, obéissant à des programmes formés par des suites d'opérations arithmétiques et logiques."

Qui dit "traitement de l'information", dit donc données à manipuler. Un programme "passe" donc son temps à traiter des données. Pour pouvoir traiter ces données, l'ordinateur doit les ranger dans sa mémoire (RAM - Random Access Memory). La RAM se compose de cases dans lesquelles nous allons ranger ces données (une donnée dans une case). Chaque case a une adresse (ce qui permet au processeur de savoir où sont rangées les données).

Alors, qu'est-ce qu'une variable ?

Eh bien, c'est une petite information (une donnée) temporaire que l'on stocke dans une case de la RAM. On dit qu'elle est "variable", car c'est une valeur qui peut changer pendant le déroulement du programme.

Une variable est constituée de 2 choses :

- Elle a une valeur : c'est la donnée qu'elle "stocke" (par exemple le nombre entier 5)
- Elle a un nom : c'est ce qui permet de la reconnaître. Nous n'aurons pas à retenir l'adresse de mémoire, nous allons juste indiquer des noms de variables à la place.

```
a = 5
```

Grâce à cette ligne, nous avons défini une variable qui porte le nom **a** et qui "contient" **le nombre entier 5**. Plus précisément, nous dirons que la **variable a** référence **le nombre entier 5**.

À faire vous-même 2

Dans la partie "éditeur" de Pyzo, saisissez le code suivant :

```
point_de_vie = 15
```

Après avoir exécuté le programme en cliquant sur " exécuter " puis " démarrer ... ", il est possible de connaître la valeur référencée par une variable en utilisant la partie "console".

Dans le cas qui nous intéresse ici, tapez point_de_vie dans la console.

Après avoir appuyé sur la touche "Entrée", vous devriez voir la valeur référencée par la variable point_de_vie s'afficher dans la console. Écrivez les deux lignes :

- **>>> point_de_vie**
- **15**

N.B. : Dans la suite la procédure sera toujours la même :

- Vous utiliserez la partie "éditeur" pour saisir votre programme
- vous utiliserez la partie "console" pour afficher la valeur référencée par une variable

À faire vous-même 3

Écrire un programme dans lequel on attribue la valeur 12 à la variable point_de_force. La valeur référencée par la variable point_de_force devra ensuite être affichée dans la console.

Nous venons de voir qu'une variable peut référencer un nombre entier, mais elle peut aussi référencer un nombre à virgule :

```
i = 5.2
```

Prenez bien garde, nous utilisons un point à la place d'une virgule (convention anglo-saxonne).

Une variable peut donc référencer plusieurs types d'entités (pour l'instant nous n'en avons vu que deux, mais nous en verrons d'autres plus loin) : les nombres entiers ("integer" en anglais, abrégé en "int") et les nombres à virgule ("float" en anglais). Il est possible de connaître le type de l'entité référencé par une variable à l'aide de l'instruction "type".

À faire vous-même 4

Testez le programme suivant :

```
a = 5.2
b = 12
```

Tapez type(a) puis type(b) dans la console . Écrivez les différentes lignes de la console.

>>> type(a)

<class 'float'>

>>> type(b)

<class 'int'>

Comme vous pouvez le constater, le type de la grandeur référencée par la variable a et le type de la grandeur référencée par la variable b s'affichent dans la console

3 - Un peu de calculs

Un ordinateur est bien évidemment capable d'effectuer des opérations mathématiques (arithmétiques).

Les signes utilisés sont classiques : +, - , * (multiplication), / (division), // (division euclidienne) ou encore % (modulo : reste d'une division euclidienne).

Il est tout à fait possible d'effectuer des opérations directement avec des nombres, mais il est aussi possible d'utiliser des variables.

→ Avant de continuer, chargez le document " exercices 1ère partie " et complétez l'exercice 1.

À faire vous-même 5

Essayez d'écrire un programme qui additionnera le contenu de 2 variables (nom des variables : a et b). Le résultat de cette opération devra être référencé par une troisième variable nommée resultat (attention pas d'accent dans les noms de variable). Testez votre programme en utilisant la console pour vérifier la valeur référencée par la variable resultat.

À faire vous-même 6

D'après vous, que fait ce programme ?

```
a = 11
a = a + 1
```

Réponse écrite.

Il additionne 1 à a et met cette valeur dans la variable a ;
résultat : a = 12

Vérifiez votre réponse en exécutant le Programme (utilisation dans console pour déterminer la valeur référencée par la variable a à la fin du programme)

Détaillons ce qui se passe dans le "À faire vous-même 6":

- nous créons une variable a qui référence l'entier 11
- nous affichons à l'écran la valeur référencée par a (c'est-à-dire 11)

La suite est un peu plus complexe, mais très importante à comprendre. Il va falloir lire la ligne "a = a + 1" de droite à gauche, décortiquons cette ligne :

- "a + 1" : nous prenons la valeur actuelle de a (c'est-à-dire 11) et nous ajoutons 1 à 11, à droite de l'égalité nous avons donc maintenant la valeur 12
- nous attribuons la valeur qui vient d'être calculée à la variable a
- nous affichons à l'écran la nouvelle valeur référencée par a

Ce raisonnement peut être généralisé pour éviter des erreurs parfois difficiles à corriger : dans une égalité, commencer toujours par évaluer l'expression se trouvant à droite du signe égal.

4 - Exposant, racine carrée, fonctions trigonométriques

Il est aussi possible d'effectuer des calculs plus complexes en utilisant par exemple des exposants, des racines carrées, des fonctions trigonométriques… Sur le document " exercices 1ère partie " complétez l'exercice 2, le premier tableau.

Pour utiliser ces fonctions mathématiques plus avancées, il est nécessaire d'ajouter une ligne au début de votre programme :

```
import math
```

Cette ligne permet d'importer (et donc d'utiliser) le module "math" (ce module contient toutes les fonctions mathématiques "classiques").

Voici quelques exemples :

- math.pow(x,a) permet de calculer x à la puissance a **!!!!!! Eh non !!**
- math.cos(x) permet de calculer le cosinus de l'angle x (l'angle x doit être en radian) (nous avons la même chose pour le sinus ou la tangente)
- math.sqrt(x) permet de calculer la racine carrée de x

Si vous avez besoin d'autres fonctions mathématiques, je vous invite à consulter la documentation de Python : https://docs.python.org/3/library/math.html

Vous pouvez compléter maintenant dans l'exercice 2 les deux autres tableaux.

À faire vous-même 7

Quelles sont les valeurs référencées par les variables de d à i après l'exécution du programme suivant :

```
import math
a = 5
b = 16
c = 3.14 / 2
d = b / a
e = b // a
f = b % a
g = math.pow(a,2)
h = math.sqrt(b)
i = math.sin(c)
```

Remplir le tableau.

Variables	Valeurs référencées après exécution du programme	Commentaires si nécessaire
a = 5	**5**	**Affectation**
b = 16	**16**	**Affectation**
c = 3.14 / 2	**1.57**	**Quotient décimal**
d = b / a	**3.2**	**Quotient décimal**
e = b // a	**3**	**Quotient entier division euclidienne**
f = b % a	**1**	**Reste division euclidienne**
g = math.pow(a,2)	**25**	**Puissance ici de 2**
h = math.sqrt(b)	**4**	**Racine carré**
`i =math.sin(c)`	**0.9999996829318346**	**16 décimales**

Vérifiez vos réponses à l'aide de la console

À noter qu'il est tout à fait possible de "mélanger" des nombres entiers et des nombres à virgules ("3.14 / 2") dans une opération.

À faire vous-même 8

Écrire un programme permettant de répondre à la question suivante : "Quel est le type du résultat d'une addition d'un integer et d'un float ?" Réponse : **c'est un float.**

5 - Chaînes de caractères

Les variables peuvent aussi référencer des suites de caractères, que l'on appelle "chaîne de caractères".

À faire vous-même 9

Tester le code suivant :

```
ma_chaine = "Bonjour le monde !"
```

Vérifiez que la variable ma_chaine référence la chaîne de caractères "Bonjour le monde !", vérifier son type et notez-le. **< class 'str' >**

6 - Le signe + et les chaînes de caractères

L'utilisation du signe + ne se limite pas à l'addition. Il est aussi utilisé pour la **concaténation**.

D'après Wikipédia :

« Le terme concaténation (substantif féminin), du latin cum («avec») et catena(«chaîne, liaison»), désigne l'action de mettre bout à bout au moins deux chaînes. »

Comme vous avez pu le deviner en lisant la définition ci-dessus, la concaténation va concerner les chaînes de caractères.

À faire vous-même 10

Quelle est la chaîne de caractère référencée par la variable mon_expression après l'exécution du programme ci-dessous ? Réponse : **HelloWorld (sans espace)**

Validez votre réponse en testant ce programme.

```
a = "Hello"
b = "World"
mon_expression = a + b
```

7 - Chaînes de caractères et variables

Il est aussi possible de *concaténer* une chaîne de caractères et une ou plusieurs variables :

À faire vous-même 11

Que doit référencer res dans le programme suivant ? Réponse : **'Bonjour le monde !'**

Testez le code :

```
ma_chaine_1 = "Bonjour "
ma_chaine_2 = "le "
res = ma_chaine_1 + ma_chaine_2 + "monde!"
```

Les 2 variables ma_chaine_1 et ma_chaine_2 référencent 2 chaînes de caractères, nous avons donc bien ici une concaténation.

Mais que se passe-t-il si la variable référence un nombre (entier ou flottant) ? Réponse : **ça ne fonctionne pas**

À faire vous-même 12

Testez le code suivant :

```
mon_nombre = 5
res = "Nombre de personnes : " + mon_nombre
```

```
 Traceback (most recent call last):
File "<console>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Comme vous pouvez le constater, nous avons droit à une erreur. En effet, il n'est pas possible de concaténer une chaîne de caractères et un nombre.

Python nous offre 2 solutions :

- l'utilisation de la méthode "str"
- l'utilisation des "fstring"

La méthode (nous verrons plus loin la notion de méthode) "str" permet de transformer un nombre en chaîne de caractères (si la transformation n'est pas possible, nous aurons une erreur)

À faire vous-même 13

Testez le code suivant :

```
mon_nombre = 5
mon_nombre = str(mon_nombre)
```

Quel est le type de la valeur référencée par la variable mon_nombre après l'exécution du programme ci-dessus ? Réponse : **<class 'str'>**

À faire vous-même 14

Testez le code suivant :

```
mon_nombre = 5
res = "Nombre de personnes : "  + str(mon_nombre)
```

Tout fonctionne, car maintenant nous avons bien une concaténation entre 2 chaînes de caractères.

Les "fstring" (nouveauté de Python 3.5), permettent de résoudre ce problème de combinaison variable-chaîne de caractères.

À faire vous-même 15

Testez le code suivant :

```
mon_nombre = 5
res = f"Nombre de personnes : {mon_nombre}"
```

Notez la présence du "f" juste avant le guillemet et des accolades qui encadrent le nom de la variable. Il est possible, dans une même chaîne de caractères d'avoir plusieurs noms de variable.

Sur le document " exercices 1ère partie " vous allez pouvoir compléter le tableau de l'exercice 3 sur les nombres et les chaînes de caractères et découvrir de nouvelles fonctions.

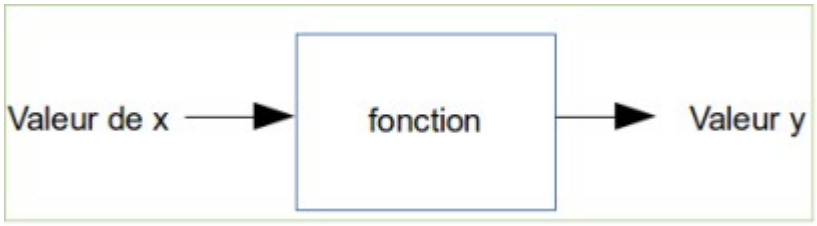
Vous pouvez charger le document " exercices 2ème partie " et faire les trois premiers exercices.

Parler de input. Et du doc mémo mis en ligne.

8 - Les fonctions

Les fonctions permettent de décomposer un programme complexe en une série de sous-programmes plus simples. De plus, les fonctions sont réutilisables : si nous disposons d'une fonction capable de calculer une racine carrée, par exemple, nous pouvons l'utiliser un peu partout dans notre programme sans avoir à la réécrire à chaque fois (on parle de factorisation du code)

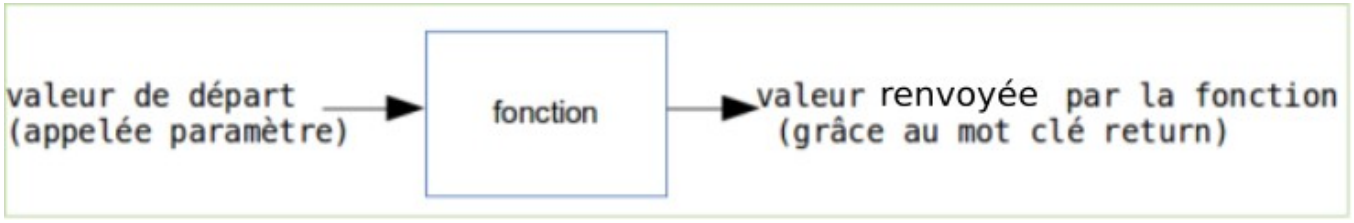
La notion de fonction en informatique est comparable à la notion de fonction en mathématiques.



Si nous avons y = 3x+2, pour une valeur donnée de x, nous aurons une valeur de y.

Exemple : x=4 donc y= 14, (y = 3*4+2=14, attention ici le signe * correspond au signe "multiplié").

La fonction en informatique est basée sur la même idée :



Voici la syntaxe employée en Python pour définir une fonction :

```
def nom_de_la_fonction(parametre):
        instruction_1
        instruction_2
        return y
suite programme
```

La fonction renvoie la valeur contenue dans la variable y.

ATTENTION : Notez bien la présence du décalage entre la première ligne et les lignes suivantes. Ce décalage est appelé indentation, l'indentation permet de définir un bloc de code. Dans l'exemple ci-dessus, l'indentation nous permet de savoir que "instruction_1", "instruction_2" et "return y" constituent un bloc de code, ce bloc correspond au contenu de la fonction. "suite programme" ne fait pas partie de la fonction, car il n'est pas indenté. Pour indenter du code, il y a 2 solutions : mettre 4 espaces ou utiliser une tabulation. En Python il est conseillé d'utiliser les 4 espaces, mais ce n'est pas une obligation. Une chose est sûre, une fois que vous avez choisi une méthode, n'en changé surtout pas au cours d'un même programme !

Codons notre exemple (y=3x+2) en créant une fonction ma_fonction :

```
def ma_fonction(x):
        y = 3 * x + 2
        return y
```

Pour "utiliser" la fonction ma_fonction, il suffit d'écrire : ma_fonction (4) (dans ce cas précis, notre fonction renverra le nombre 14).

À faire vous-même 16

Testez le programme suivant (quelle est la valeur référencée par la variable solution après l'exécution du programme) : Réponse : **20.6**

```
def ma_fonction(x):
        y = 3 * x + 2
        return y
solution = ma_fonction(6.2)
```

À faire vous-même 17

Codez en Python la fonction $y = x^2+2x+10$

Testez pour x = 3, écrivez la réponse y = **25**…… et vérifiez par le programme.

Il est possible d'écrire une fonction dans l'éditeur et d'utiliser la console (après exécution du programme) pour obtenir la valeur renvoyée par une fonction.

Il est possible de faire passer plusieurs paramètres à une fonction.

À faire vous-même 18

Quel est le résultat renvoyé par la fonction ci-dessous si l'on saisit dans la console une_autre_fonction(5, 3) ? Réponse : **18**

```
def une_autre_fonction(x, b):
        y = 3 * x + b
        return y
```

Les paramètres peuvent être des chaînes de caractères (ainsi que la valeur retournée)

À faire vous-même 19

Quel est le résultat attendu après l'exécution du programme ci-dessous et la saisie dans la console de "dit_bonjour("toto", 14)" ? Réponse : **'Bonjour toto, vous avez 14 ans.'**

```
def dit_bonjour(nom, age):
        phrase = f"Bonjour {nom}, vous avez {age} ans."
        return phrase
```

Attention : remarquez bien les guillemets autour du paramètre "toto" (c'est une chaîne de caractères).

Les paramètres ne sont pas obligatoires.

À faire vous-même 20

Testez la fonction suivante :

```
def ma_fon():
        return "voici une fonction qui ne sert à rien"
```

Il faut aussi savoir qu'une fonction ne renvoie pas forcément de valeur (le mot clé return n'est pas obligatoire). Mais si elle ne renvoie pas de valeur, que fait-elle ? Elle peut faire plein de choses, par exemple elle peut tout simplement afficher une chaîne de caractères à l'aide d'un "print". Sachez que dans certains langages, on utilise les termes méthode ou procédure pour qualifier une fonction "qui ne renvoie rien".

À faire vous-même 21

Soit le programme suivant :

```
def dit_bonjour(nom, age):
        phrase = f"Bonjour {nom}, vous avez {age} ans."
        print(phrase)
```

Testez la fonction dit_bonjour à l'aide de la console (avec par exemple un dit_bonjour("toto", 14))

Vous pouvez refaire les deux premiers exercices du document " exercices 2ème partie " en utilisant les fonctions et la fonction input

9 - Les expressions et les booléens

Si quelqu'un vous dit que "4 est égal à 5", vous lui répondez quoi ? "c'est faux". Si maintenant la même personne vous dit que "7 est égal à 7", vous lui répondrez bien évidemment que "c'est vrai".

En Python, ces deux «affirmations» ("4 est égal à 5" et "7 est égal à 7") s'écriront "4 == 5" et "7 == 7" (notez bien le double signe égal).

À faire vous-même 22

Dans la console, tapez :

```
4 == 5
```

En Python, "4 == 5" est appelé une expression, une expression est soit vraie ("True"), soit fausse ("False").

Pour l'instant nous avons vu deux grands types de données : les nombres (entier ou flottant) et les chaînes de caractères, il existe un troisième type tout aussi important que les deux premiers : les booléens. Un booléen est un type de données qui ne peut prendre que deux valeurs : vrai ("True") ou faux ("False"). Une expression (comme par exemple "4 == 5") est soit True, soit False.

ATTENTION : notez le double égal "==" qui permet de distinguer une expression et une affectation (référencement d'une valeur par une variable). Le fait de confondre le "simple égal" et le "double égal" est une erreur classique qu'il faut éviter.

À faire vous-même 23

Soit le programme suivant :

```
a = 4
b = 7
```

Quel est le résultat attendu après l'exécution de ce programme si vous saisissez dans la console "a==b" ?

False

Il est possible d'utiliser aussi l'opérateur "différent de" != .

À faire vous-même 24

Soit le programme suivant :

```
a = 4
b = 7
```

Quel est le résultat attendu après l'exécution de ce programme si vous saisissez dans la console "a != b" ?

True

Notez aussi l'existence des opérateurs :

- "strictement inférieur à" <
- "strictement supérieur à" >
- "inférieur ou égal à" <=
- "supérieur ou égal à" >=

À chaque fois ces opérateurs sont True (vrai) ou False (faux).

À faire vous-même 25

Soit le programme suivant :

```
a = 4
b = 7
```

Quel est le résultat attendu après l'exécution de ce programme si vous saisissez dans la console "a <= b", puis " a > b " ?

True

False

10 - Les conditions si - sinon

Nous allons maintenant étudier une structure fondamentale en programmation le « si ..... alors.....sinon.....».

L'idée de base est la suivante :

```
si expression:
        suite_instruction1
sinon:
        suite_instruction2
```

Si "expression" est True alors "suite_instruction1" est exécuté et "suite_instruction2" est ignoré.

Sinon (sous-entendu que "expression" est False) "suite_instruction2" est exécuté et "suite_instruction1" est ignoré.

Notez l'indentation «suite_instruction1» et de « suite_instruction2»

À faire vous-même 26

Soit le programme suivant :

```
a = 4
b = 7
if a < b:
        print("Je suis toto.");
        print("Je n'aime pas titi.")
else:
        print("Je suis titi.")
        print("Je n'aime pas toto.")
print("En revanche, j'aime le Python.")
```

Quel est le résultat attendu après l'exécution de ce programme ?

je suis toto.

Je n'aime pas titi.

En revanche, j'aime le Python.

Vérifiez votre hypothèse en testant le programme.

Vous pouvez faire les exercices 4, 5 *du document " exercices 2ème partie ".*

À faire vous-même 27

Écrire une fonction qui prend en paramètre un age. Si age est supérieur ou égal à 18 ans, la fonction devra renvoyer la chaîne de caractères "Bonjour, vous êtes majeur.". Si age est inférieur à 18 ans, la fonction devra renvoyer "Bonjour, tu es mineur."

```
def majeur(age):
        if age < 18:
                print("Bonjour, tu es mineur.")
        else:
                print("Bonjour, vous êtes majeur.")


majeur(56)
majeur(17)
```

À faire vous-même 28

Soit le programme suivant :

```
def annonce(num, prov, dest):
        if dest != "0":
```

```
                msg = f"le train n° {num} en provenance de {prov} et à
destination de {dest}, entre en gare."
        else:
                msg = f"le train n° {num} en provenance de {prov} entre en gare.
Ce train est terminus Triffouillis-les-Oies."
        return msg
```

Quel est le résultat attendu après l'exécution de ce programme si vous saisissez dans la console "annonce("4557", "Paris", "Marseille")" ? Et si vous saisissez dans la console "annonce("5768", "Bonneville", "0")" ? Vérifiez votre réponse en testant ce programme.

'le train n° 4557 en provenance de Paris et à destination de Marseille, entre en gare.'

'le train n° 5768 en provenance de Bonneville entre en gare. Ce train est terminus Triffouillis-les-Oies.'

11 - Le "ou" et le "et"

Un if peut contenir plusieurs conditions, nous aurons alors une structure de la forme :

```
si expression1 op_logique expression2:
        suite_instruction1
sinon:
        suite_instruction2
```

« op_logique » étant un opérateur logique.

Nous allons étudier 2 opérateurs logiques : le "ou" (noté en Python "or") et le "et" (noté en Python "and").

Par exemple (expression1 or expression2) est vrai si expression1 est vraie et expression2 est vraie.

Autre exemple (expression1 and expression2) est faux si expression1 est vraie et expression2 est faux.

Les résultats peuvent être regroupés dans ce que l'on appelle une table de vérité :

table de vérité pour le "ou"

expression1	**expression2**	**expression1 or expression2**
vrai	vrai	vrai
vrai	faux	vrai
faux	vrai	vrai
faux	faux	faux

table de vérité pour le "et"

expression1	**expression2**	**expression1 and expression2**
vrai	vrai	vrai
vrai	faux	faux
faux	vrai	faux
faux	faux	faux

À faire vous-même 29

Soit le programme suivant :

```
a = 5
b = 10
```

```
if a > 5 and b == 10:
        print ("Toto")
else:
        print("Titi")
if a > 5 or b == 10:
        print ("Tata")
else:
        print("Tutu")
```

Quel est le résultat attendu après l'exécution de ce programme ?

Titi

Tata

Vérifiez votre réponse en testant ce programme.

12 - Les conditions si - sinon-si - sinon

Voici une structure présentant plusieurs alternatives :

```
si condition1:
        instruction1
sinon si condition2:
        instruction2
sinon:
        instruction3
```

ceci se traduit par : `if - elif - else`

À faire vous-même 30

Créer une fonction qui prend un nombre entier en paramètre (argument) et retourne une réponse du type " nombre négatif, positif ou nul ".

```
def signe(x):
        if x ==0:
                print("nombre nul.")
        elif x > 0:
                print("nombre positif.")
        else:
                print("nombre négatif")
signe(7)
```

Vous pouvez faire l'exercice 6 du document *" exercices 2ème partie ".*

13 - La boucle while

La notion de boucle est fondamentale en informatique. Une boucle permet d'exécuter plusieurs fois des instructions qui ne sont présentes qu'une seule fois dans le code.

La structure de la boucle while est la suivante :

```
while expression:
        instruction1
        instruction2
suite programme
```

Tant que l'expression s'évalue à "True", les instructions à l'intérieur du bloc (partie indentée) seront exécutées.

À faire vous-même 31

Soit le programme suivant :

```
i = 0
while i < 10:
        print(f"i vaut : {i}")
        i = i + 1
print("C'est terminé.")
```

```
Autre façon d'écrire la même chose :
i = 0
while i < 10:
       print("i vaut : ",i)
       i = i + 1
print("C'est terminé.")
```

Quel est le résultat attendu après l'exécution de ce programme ?

i vaut : 0
i vaut : 1
i vaut : 2
i vaut : 3
i vaut : 4
i vaut : 5
i vaut : 6
i vaut : 7
i vaut : 8
i vaut : 9
C'est terminé.

Vérifiez votre réponse en testant le programme

À faire vous-même 32

Un exercice permettant de créer "un générateur automatique de punition" :

Écrire une fonction qui prendra 2 paramètres : une chaîne de caractère et un nombre entier

Par exemple :

Si on passe comme paramètres à notre fonction : "Je ne dois pas discuter en classe" et 3

La fonction devra permettre d'afficher :

Je ne dois pas discuter en classe

Je ne dois pas discuter en classe

Je ne dois pas discuter en classe

```
def punition(text,n):
        n = 0
        while n <3:
                print(text)
                n=n+1

punition("Je ne dois pas discuter en classe.", 3)
```

À faire vous-même 33

Écrire une fonction permettant d'afficher une table de multiplication. Cette fonction devra prendre en paramètre la table désirée.

Par exemple si l'on passe le paramètre 3 à la fonction, la fonction devra permettre d'afficher :

1 x 3 = 3

2 x 3 = 6

...

...

10 x 3 = 30

```
def table(n):
    i = 1
    while i <11:
        print(i,"x",n," =",i*n )
        i = i + 1
table(3)
```

Vous pouvez faire les exercices 7 et 8 du document *" exercices 2ème partie ".*

14 – La boucle for

Il existe une autre façon pour répéter plusieurs fois la même instruction.
Testez le code suivant :

```
for i in range(3) :
    print(" Je ne dois pas discuter en classe ")
```

La boucle for s'utilise quand on connaît le nombre de tours de boucle.

À faire vous-même 34

Comme pour l'exercice 33, réaliser une fonction qui affiche une table de multiplication en utilisant une *boucle for*. La fonction prend en paramètre la table voulue.

```
def table(n):
    for i in range(11):
        print(i,"x",n," =",i*n )
table(3)
```

Vous pouvez maintenant faire tous les exercices du document *" exercices 2ème partie " !*