Séquence 1
Programmation en Python – partie 1

Le langage Python est un langage facile à apprendre avec des lignes de commande claires et concises.
La mise en place de l'environnement de développement est très simple, il n'y a pas de compilateur à mettre en place car le langage est interprété.
Python est utilisé pour le développement de nombreux logiciels Open Source. Connaitre ce langage est donc un atout.

Le langage Python tire son nom d'une émission humoristique britannique : « *Monty Python's flying Circus* ». Le créateur du langage, le mathématicien Hollandais Guido van Rossum le nomma « Python » en hommage à ce programme TV en 1991.

Cette séquence est découpée en chapitres qui introduisent chacun une notion. Lisez les chapitres dans l'ordre car ils sont liés entre eux.
Nous allons revoir les bases de la programmation en Python.
Conseils pour aborder cette séquence, n'hésitez pas à reproduire les lignes de code dans un éditeur de texte et à interpréter le code pour voir le résultat et comprendre.
Prenez le temps de lire chacune des lignes.

À la fin de la séquence se trouve les exercices. Faites-les au fur et à mesure de votre progression.

Objectifs :

- Maitriser les instructions élémentaires :
  - Connaitre les notions de variable et de fonction ;
  - Distinguer une expression et une instruction ;
  - Utiliser une affectation, une instruction conditionnelle, une boucle

Chapitres :

CHAPITRE 1 INTRODUCTION

Un Programme est la description d'un algorithme dans un langage compréhensible par un humain mais aussi et surtout par une machine.
C'est la machine qui va exécuter le programme afin de traiter les données et les instructions données.

Il existe une multitude de langage de programmation avec chacun leur particularité. Certain ont une syntaxe plus permissive que d'autres. D'autre sont plus proche du langage naturel humain ou à l'inverse plus proche du langage de la machine.

Le langage Python est de plus en plus répandu dans l'enseignement supérieur et au lycée dans le cadre de l'enseignement des mathématiques.
Ce langage a été créé par Guido Van Rossum, un ingénieur informaticien néerlandais en 1991. Il a travaillé pour Google puis Dropbox. Nous allons utiliser la version 3 de Python.

Le langage Python est dit « multiplateforme » car il fonctionne aussi bien sur des ordinateurs sous Windows, Linux, MacOs, Android ou IOS. C'est un langage gratuit et placé sous licence libre.

Les constructions élémentaires en langage Python sont communes à de nombreux autres langages de programmation.

Un programme est composé :

- De séquences (instructions exécutées l'une après l'autre dans l'ordre où elles sont écrites).
- Des définitions de variables et de fonctions.
- D'affectations de valeurs.
- D'instructions conditionnelle.
- De boucles.
- Des appels de fonctions.

Voici ce que l'on peut faire avec du Python :

- Des petits programmes très simples, appelés **scripts**, chargés d'une mission très précise sur votre ordinateur ;
- Des programmes complets, comme des jeux, des suites bureautiques, des logiciels multimédias, des clients de messagerie…
- Des projets très complexes, comme des progiciels (ensemble de plusieurs logiciels pouvant fonctionner ensemble, principalement utilisés dans le monde professionnel).

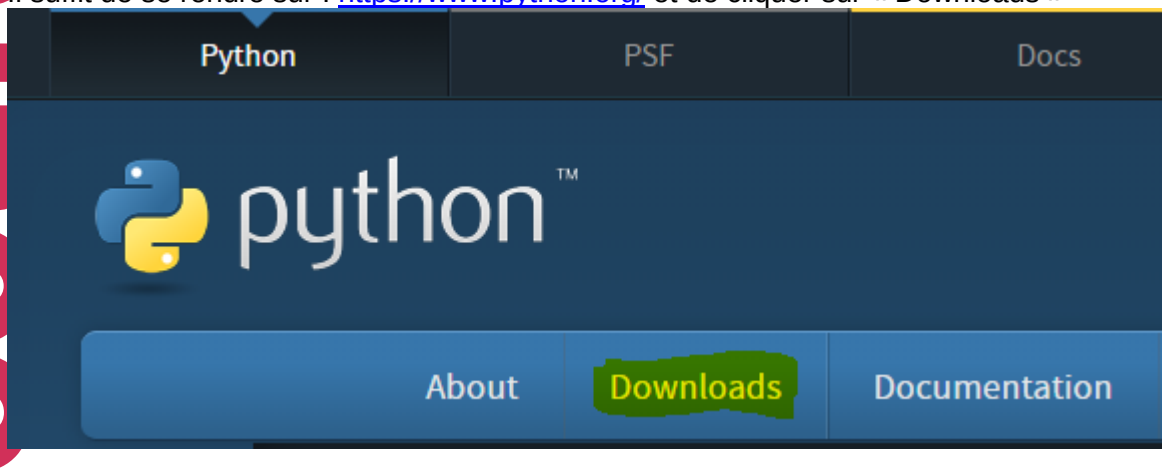
Python est un langage de programmation **interprété**, c'est-à-dire que les instructions que vous lui envoyez sont « transcrites » en langage machine au fur et à mesure de leur lecture. D'autres langages (comme le C / C++) sont appelés « langages **compilés** » car, avant de pouvoir les exécuter, un logiciel spécialisé se charge de transformer le code du programme en langage machine. On appelle cette étape la « **compilation** ». À chaque modification du code, il faut rappeler une étape de compilation.

Les avantages d'un langage interprété sont la simplicité (on ne passe pas par une étape de compilation avant d'exécuter son programme) et la portabilité (un langage tel que Python est censé fonctionner aussi bien sous Windows que sous Linux ou Mac OS, et on ne devrait avoir à effectuer aucun changement dans le code pour le passer d'un système à l'autre). Cela ne veut pas dire que les langages compilés ne sont pas portables, loin de là ! Mais on doit utiliser des compilateurs différents et, d'un système à l'autre, certaines instructions ne sont pas compatibles, voire se comportent différemment.

En contrepartie, un langage compilé se révélera bien plus rapide qu'un langage interprété (la traduction à la volée de votre programme ralentit l'exécution), bien que cette différence tende à se faire de moins en moins sentir au fil des améliorations. De plus, il faudra installer Python sur le système d'exploitation que vous utilisez pour que l'ordinateur puisse comprendre votre code.

INSTALLER PYTHON sous WINDOWS.

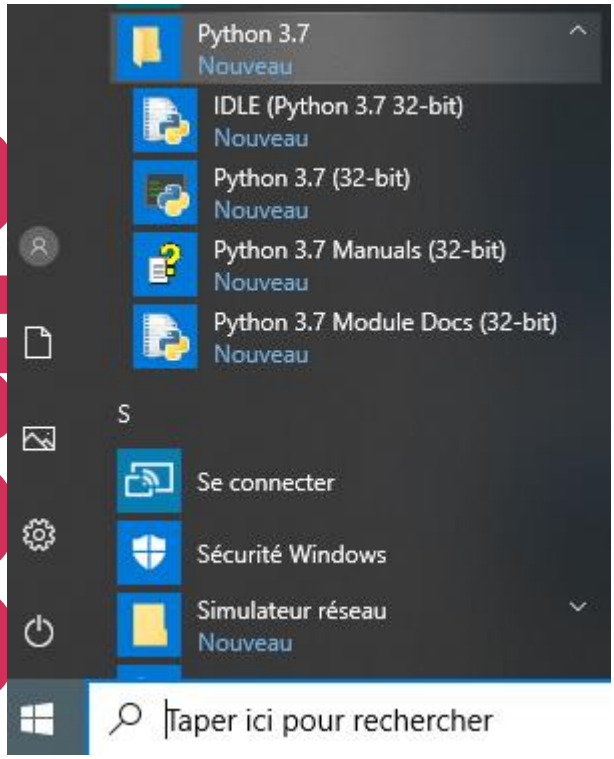
Il suffit de se rendre sur : https://www.python.org/ et de cliquer sur « Downloads »



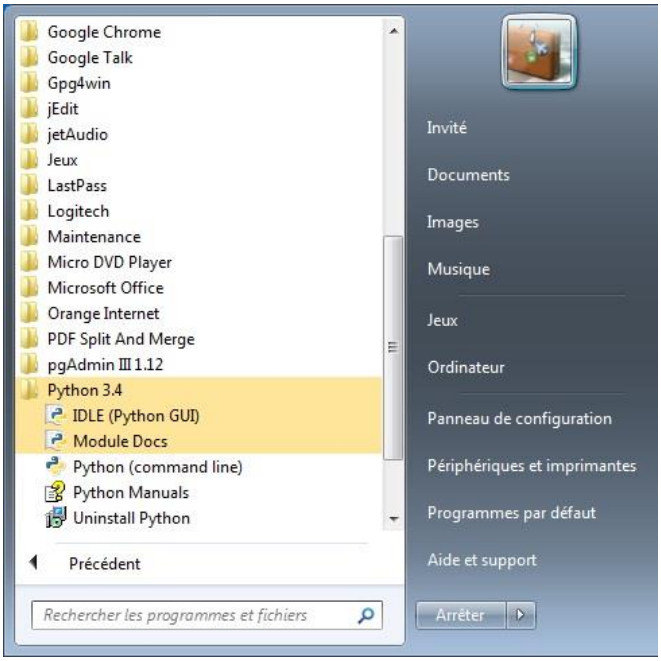
Et de télécharger la dernière version ou « release ».

Exécutez le programme

Une fois l'installation terminée, vous pouvez vous rendre dans le menu `Démarrer>Tous les programmes`. Python devrait apparaître dans cette liste.



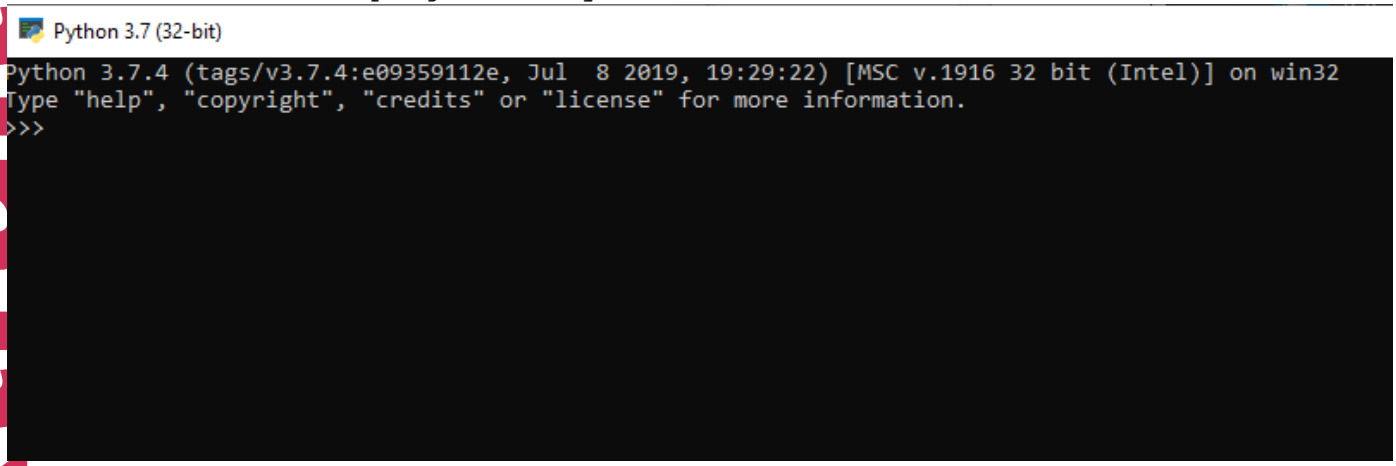
Windows 10



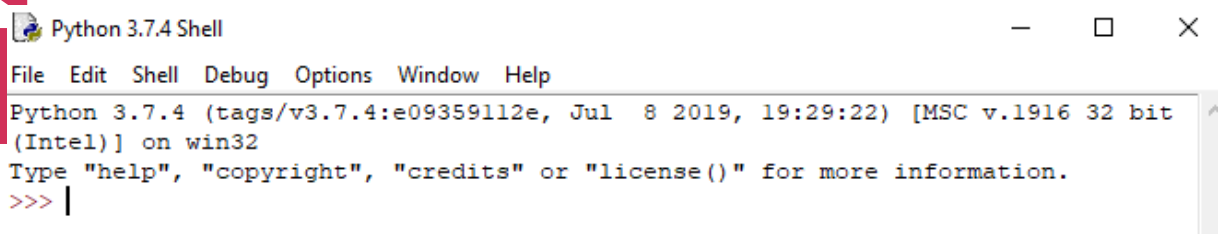
Windows 7

LANCER PYTHON  python-3.7.4.exe

Vous avez plusieurs façons d'accéder à la ligne de commande Python, la plus évidente consistant à passer par les menus `Démarrer>Tous les programmes>Python 3.7>Python (32-bit)`. Ou `Démarrer>Tous les programmes>Python 3.7>` IDLE.

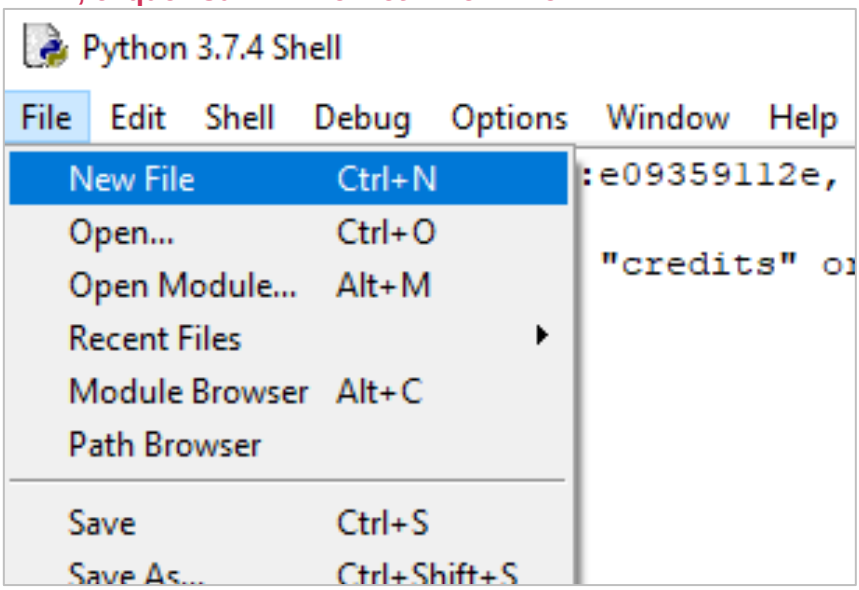


Ou via IDLE :

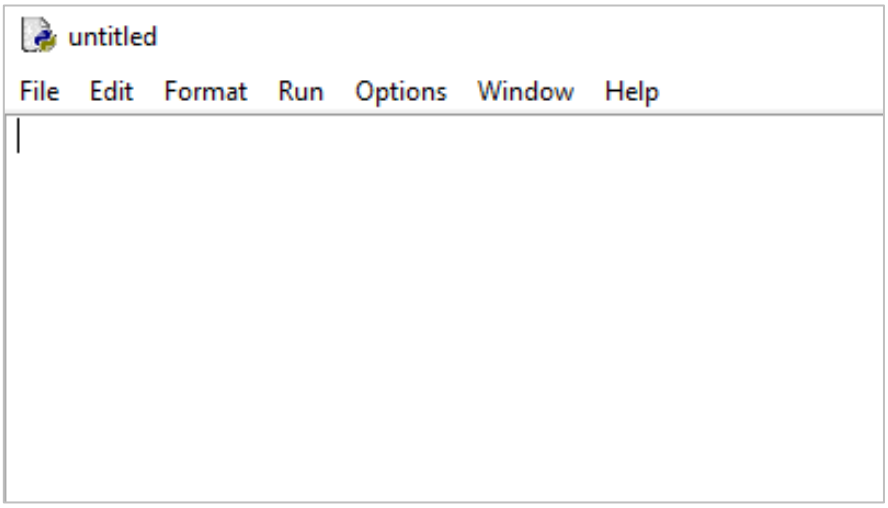


Comment tester du code ou coder avec les logiciels utilisés ?

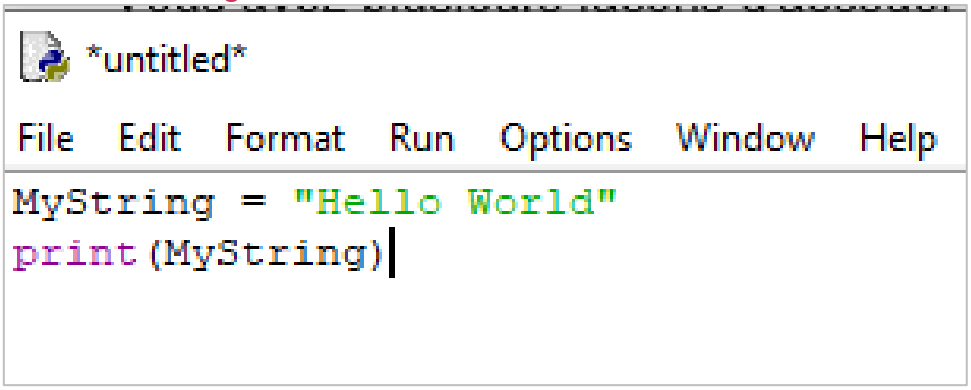
✓ Dans IDLE, cliquez sur : « File » et « New File ».



✓ Une nouvelle fenêtre s'ouvre :

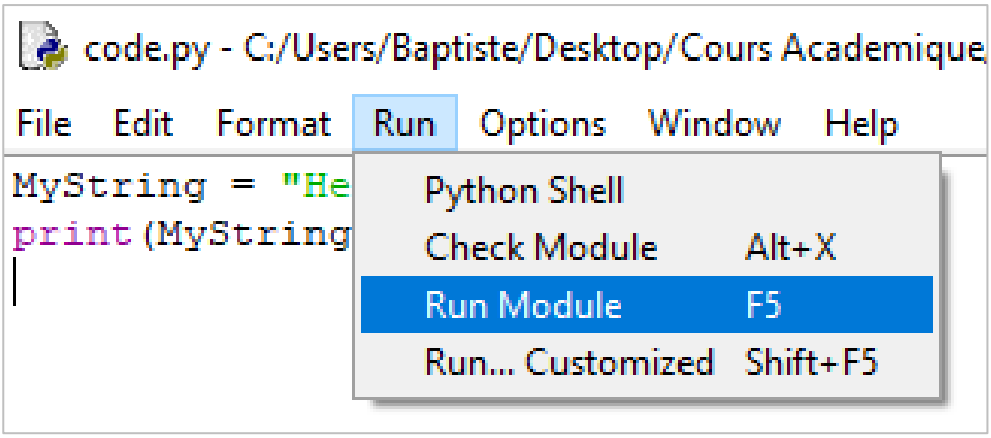


✓ Saisissez votre ligne de votre :

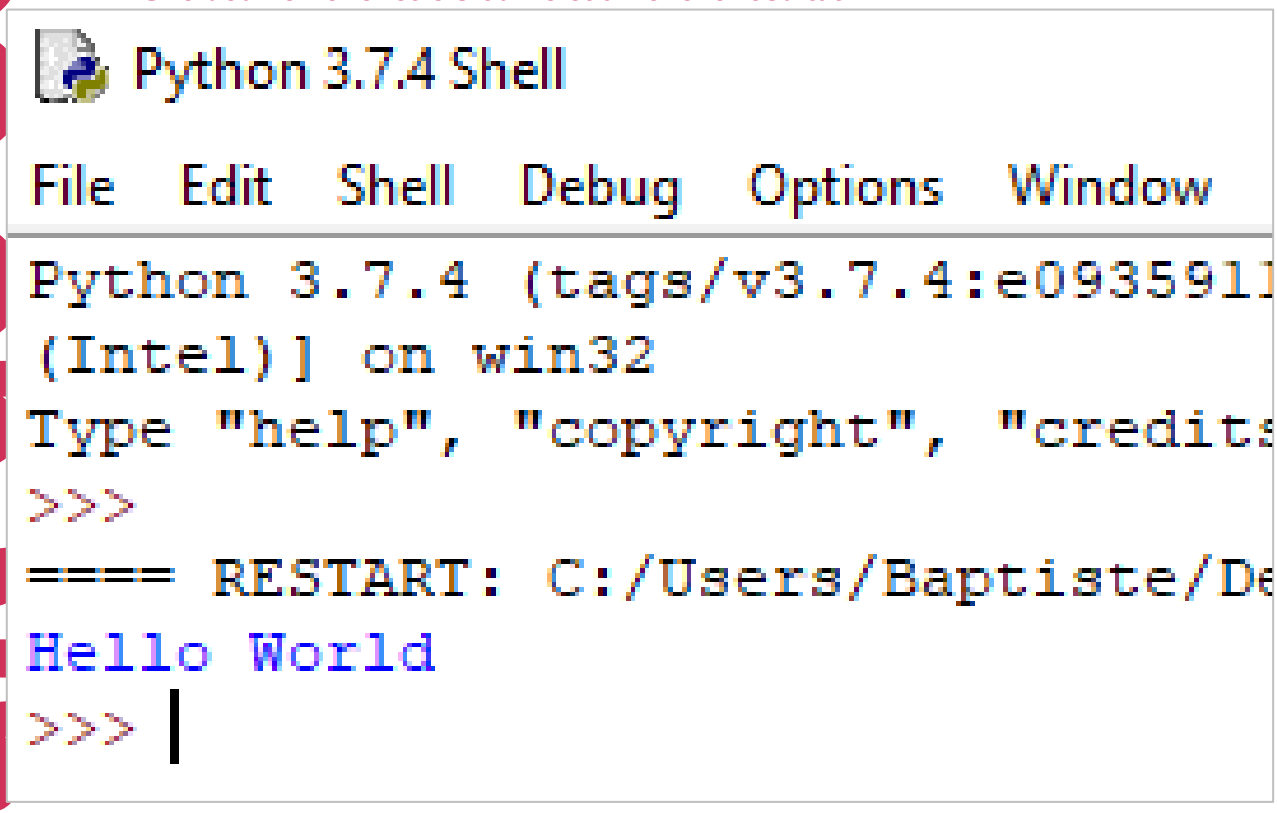


✓ Enregistrez-le

✓ Exécutez



✓ Une deuxième fenêtre s'ouvre et affiche le résultat :



CHAPITRE 2 ÉLÉMENTS DE BASE

À SAVOIR

Pour afficher du texte dans la console en Python, on utilise la fonction : **`print()`.**

Pour demander à l'utilisateur de saisir des données, on utilise la fonction : **`input()`.**
La fonction Input renvoie toujours une chaine de caractères.

1. Variables et affectation

Dans un programme, les données utilisées sont stockées dans des variables. Une affection est le fait d'associer une donnée (valeur ou expression) avec un nom.
Une variable est comme une boîte sur laquelle il y a une étiquette avec un nom et dans laquelle on y range des informations diverses.
Le nom de la variable peut être n'importe quelle chaine alphanumérique (sauf certains mots clés réservés) et ne doit pas commencer par un chiffre.
L'opérateur d'affectation est noté « = ».

Exemple 1 : Cette instruction associe la valeur 3 au nom x.

```
X=3
```

Exemple 2 : Cette instruction associe la valeur de l'expression à droite du signe « = ».

```
Y = 3 + 5
```

« y » vaut 8.

Exemple 3 :

```
X=3
Y=3+5
Z = X + Y
```

« z » vaut 11. Car il est la somme des valeurs de « x » et de « y » au nom « z ».
En python, nous pouvons faire des affectations multiples pour gagner du temps dans la rédaction et d'économiser des lignes de code.
Ainsi, ces 3 lignes d'instructions :

```
X=1
Y=3
Z=5
```

Peuvent tout simplement s'écrire :

```
x,y,z = 1,2,5
```

On pourrait aussi écrire :

```
x=1; y=2; z=5
```

DÉFINITION

Vocabulaire à connaitre :

Une variable est composée d'un nom (ou identificateur) ; d'une adresse en mémoire où est enregistrée une valeur (ou un ensemble de valeurs), d'un type qui définit ses propriétés.

Une expression a une valeur qui est le résultat d'une combinaison de variables ou d'objets, de constantes et d'opérateurs.

Une instruction est une commande qui doit être exécutée par la machine.

Une affectation est une instruction qui commande à la machine de créer une variable en lui précisant son nom et sa valeur.

Attention à ne pas confondre expression et instruction. Une expression se calcul, elle possède une valeur. L'instruction s'exécute et n'a pas de valeur.
L'écriture `x=0.5*x**2+1` est une **instruction** : affecter la valeur de **l'expression** `0.5*x**2+1`.

2. Les types simples.

Les types de base qui permettent de définir l'ensemble des valeurs qui peuvent prendre les variables sont :

- Les types numériques « **`int`** ».

Ce type représente les nombres entiers. La taille d'une variable `int` n'est limité que par la capacité de la machine et le temps nécessaire à leur utilisation.

- Les types booléens « **`bool`** ».

Ce type permet de représenter les valeurs booléennes `True` (vrai) ou 1 en binaire et `False`(faux) ou 0 en binaire.

- Les types flottant « **`float`** ».

Ce type est utilisé pour les nombres réels. La virgule est remplacée par le point.

- Les types chaines de caractères « **`str`** ».

Les chaines de caractères.

3. Opérations sur les types numériques

Nous pouvons effectuer toutes les opérations mathématiques. Voici un tableau avec le nom de l'opération et le symbole utilisé en Python pour la réaliser :

Addition	**+** Exemple : `a + b`
Soustraction	**-** Exemple : `a - b`
Multiplication	***** Exemple : `a * b`
Exponentiation	** Exemple : `a ** b`
Division	**/** Exemple : `a / b`
Division entière	**//** Exemple : `a // b`
Opération modulo	**%** Exemple : `a % b`

Pour chacune des écritures il existe des syntaxes permettant d'aller plus vite dans la rédaction comme :
`a = a + b` peut s'écrire `a += b` par exemple.

4. Comparaison et opération booléens

Les opérateurs mathématiques de comparaisons s'écrivent ainsi :

Egale	==
Différent	!=
Inférieur	<
Inférieur ou égale	<=
Supérieur	>
Supérieur ou égale	>=

Valeur de retour sur les opérations

`x==y` prend la valeur `True` si `x` et `y` sont égaux, sinon prend la valeur `False`.
`x!=y` prend la valeur `True` si `x` et `y` sont différent, sinon prend la valeur `False`.

Valeur de retour sur les opérations logiques

`a and b` prend la valeur `True` si `a` et `b` sont `True` et sinon prend la valeur `False`.
`a or b` prend la valeur False si `a` et `b` sont `False` et sinon prend la valeur `True`.
`not a` prend la valeur `True` si `a` est `False` et prend la valeur `False` si `a` est `True`.

5. Le type `str` ou chaine de caractères

`str` est une abréviation de string en anglais qui veut dire chaine de caractères.

Une chaine de caractère est par exemple tout ce que l'on saisit avec les touches du clavier. On utilise des guillemets ou des apostrophes pour les déclarer.

```
MyString = "hello world!"
MyString2 = 'hello world!'
```

Si vous écrivez `Mystring = Hello, Hello` sera considéré comme étant une variable. Si elle n'existe pas, il y aura une erreur. Si elle existe, `MyString` aura alors la valeur de la valeur de `Hello`.

Nous pouvons connaitre la longueur d'une chaine, qui est le nombre de caractère qui la compose, grâce à la fonction `len`.

```
len("hello") #a pour valeur un entier : 5
MyString = "Hello world"
len(MyString) #a pour valeur un entier : 11
```

Chaque caractère de la chaine possède un indice qui commence de 0 à (longueur de la chaine -1).

Indice : 0	Indice : 1	Indice : 2	Indice : 3	Indice : 4	Indice : 5	Indice : 6	Indice : 7	Indice : 8	Indice : 9
H	E	L	L	O		Y	O	U	!

Nous avons ici la chaine "HELLO YOU !".
Si nous faisons un : `len("HELLO YOU !")` nous obtenons la longueur 10. Toutefois nous voyons dans le tableau que le dernier caractère se situe à l'indice 9(10 – 1).

Il est possible d'accéder à l'indice `i` d'une chaine grâce à cette syntaxe :

```
MyString = "Hello world"
MyString[2] #Nous accédons à l'indice 2 de la chaine. Soit au caractère l
```

Nous pouvons avoir accès également à une suite de caractères d'une chaine avec la notation : `MyString [i : j]`. L'indice `i` est inclus et `j` est exclu.

6. Les types composés

```
MyString = "Hello world"
MyString [2 :4] #résultat:  ll
```

Il s'agit des types `tuple`, `list`, `dict`.

Nous étudierons ici les `list`.
Un objet de type **`list`** (« une liste »), représente un ensemble ordonné d'objets éventuellement de types différents.
De la même manière qu'avec les chaines de caractères, les éléments de la liste sont ordonnés en commençant à l'indice `0`.

Comment déclarer une liste :

```
MyList1= [] #une liste vide.
MyList2=[4] #une liste avec un seul élément, ici l'entier 4.
MyList3=[5, 'hello', 3.14, ['a', 'b']] #une liste avec des éléments de
différents types.
```

La fonction `len` est aussi utilisable avec les listes et renvoie sa longueur. `len(MyList3)` a pour valeur l'entier 4.

L'accès à un élément ou une suite d'éléments se fait comme pour les chaînes :

```
MyList3[1] #est l'élément 'hello'
MyList3[0:3]  #est  la  liste  des  éléments  d'indices  0,1,2  soit
[5,'hello',3.14]
```

Il est possible de modifier le contenu d'une liste de la sorte :

```
MyList3[1] = "AU REVOIR"
MyList3 #vaut alors [5, 'AU REVOIR', 3.14, ['a', 'b']]
```

La méthode `append` permet d'ajouter des éléments en fin de liste.

```
MyList.append("ELEMENT1")
MyList.append("ELEMENT2")
#MyList vaut: ["ELEMENT1", "ELEMENT2"]
```

7. Opération sur les types `str` et `list`

- Il est possible d'indexer à partir du dernier élément :

Exemple : Nous avons une liste de : 7 éléments, rangés de l'index 0 à 6.

```
MyList["item1","item2","item3","item4","item5","item6","item7"]

#Je peux directement aller au dernier élément :
MyList[-1]
#Et on en déduit comment accès à l'avant dernier :
MyList[-2]
```

- La concaténation.

```
Str1 = "BON"
Str2="JOUR"
```

Je peux créer une nouvelle chaine qui sera l'assemblage de `Str1` et `Str2`, on appelle cela faire une **concaténation**.

```
Str3 = Str1 + Str2
```

`Str3` vaut alors « **BONJOUR** »

- Nous pouvons aussi effectuer la concaténation de n copies :

```
strConcat = 3*Str3
```

`strConcat` vaut « **BONJOURBONJOUR** »

- Nous pouvons changer le type des variables grâce aux fonctions : `int`, `float`, `str`

```
StrPI = "3.1415"
```

`StrPI` est de type `string`. Pour la convertir en `float` il suffira d'écrire ceci :

```
float(StrPI)
```

De ce fait :

```
NombrePI = float(StrPI)
```

`NombrePI` sera de type `float`.

- La fonction `list` permet de convertir une chaine de caractère en une liste dont les éléments sont les différents caractères de la chaine.

```
String = "3.1415"
MyList = list(String)
```

`MyList vaut donc ['3', '.', '1', '4', '1', '5']`

CHAPITRE 3 INSTRUCTIONS CONDITIONNELLES ET BOUCLES

L'indentation est le décalage vers la droite du début de ligne.
C'est un élément très important de la syntaxe en Python mais aussi dans tous les autres langages.

Cela permet de délimiter visuellement des blocs de code et aide à la lisibilité.
En Python, quand vous créez un bloc, la ligne précédente l'indentation se termine par le signe « : ».

Instructions conditionnelles

```
if condition:
    Instructions
```

Condition désigne une expression et instructions désigne une instruction ou un bloc d'instructions écrites sur plusieurs lignes.

Exemple :

```
if n == 4:
    n = 4 * 2
```

Nous pouvons aussi ajouter des instructions si la condition n'est pas respectée avec le mot clé `else`

```
if n == 4:
    n = 4 * 2
else:
    n = 4 + 1
```

Nous pouvons aussi ajouter une série de conditions dans le cas où les précédentes ne seraient pas respectées avec le mot clé `elif`.

```
if degree <= 0:
    message = "Il géle !!"
elif degree > 0 and degree < 20:
    message = "Il fait froid !!"
else:
    message ="Il faut chaud!!"
```

N'oubliez pas que c'est l'indentation qui permet de délimiter les blocs d'instructions à exécuter si la condition est vérifiée.

Boucles conditionnelles

Structure :

```
while condition:
    instructions
```

Tant que la condition est respectée, alors les instructions seront exécutées.

```
a = 10
while a > 0:
    print("a est supérieur à zéro")
    a = a - 1
```

Si j'exécute le script, j'obtiens à l'écran :

```
a est supérieur à zéro
a est supérieur à zéro
a est supérieur à zéro
a est supérieur à zéro
a est supérieur à zéro
a est supérieur à zéro
a est supérieur à zéro
a est supérieur à zéro
a est supérieur à zéro
a est supérieur à zéro
```

Tant que `a > 10` est `True`, alors on affiche « `a est supérieur à zéro` » puis on soustrait 1 à la variable a.
Au bout de 10 « tours » dans la boucle, a se retrouve égale à zéro, la condition `a > 0` n'est plus vrai. Les instructions cessent alors d'être exécutées.

Boucles non conditionnelles

Structure :

```
for i in range(n):
    instructions
```

Cette boucle permet de répéter n fois une instruction ou un bloc d'instructions.

```
for car in "bonjour":
    print(10*car)
```

Exécution :

```
bbbbbbbbbb
oooooooooo
nnnnnnnnnn
jjjjjjjjjj
oooooooooo
uuuuuuuuuu
rrrrrrrrrr
```

Explication :
Pour chaque caractère un à un de la chaine « bonjour », on affiche 10 fois le caractère.

```
for i in range(10):
    print("hello")
```

Exécution : Pour chaque valeur de i en partant de 1, puis en augmentant de 1 jusqu'à 10, afficher : hello.

```
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
```

```
for i in range(6,10):
    print("hello")
```

Exécution : Pour chaque valeur de i en partant de 6, puis en augmentant de 1 jusqu'à 10, afficher : hello.

```
hello
hello
hello
hello
```

```
for i in range(6,10,2):
    print("hello")
```

Exécution : Pour chaque valeur de i en partant de 6, puis en augmentant de 2 jusqu'à 10, afficher : hello.

```
hello
hello
```

Ajouter l'instruction **`break`** dans la boucle permet de sortir de celle-ci. L'instruction **`continue`** permet d'éviter un passage dans la boucle.

Modifier une liste :

Déjà affichons le contenu d'une liste :

```
MyList = ['bijou','caillou','chou','genou','hibou','joujou','pou']
for i in range(len(MyList)):
    print(MyList[i])
```

Exécution :

```
bijou
caillou
chou
genou
hibou
joujou
pou
```

Nous souhaitons ajouter un "x" à la fin de chaque mot, et les afficher de nouveau.

```
MyList = ['bijou','caillou','chou','genou','hibou','joujou','pou']
for i in range(len(MyList)):
    MyList[i] = MyList[i] + "x"
    print(MyList[i])
```

Exécution :

```
bijoux
cailloux
choux
genoux
hiboux
joujoux
poux
```

Séquence 1
Programmation en Python – partie 1
EXERCICES

EXERCICE 1 VRAI / FAUX

	VRAI	FAUX
1. Voici les instructions suivantes : `x=3, y=5, x=y, y=x` La valeur de x est 5 et la valeur de y est 3.	☐	☐
2. Voici les instructions suivantes : `x=3, y=5, y==x, x=y` La valeur de x est 5 et la valeur de y est 5.	☐	☐
3. La valeur de `len("au revoir")` est 9.	☐	☐
4. La valeur de `"toutou"*2` est « toutoutoutou ».	☐	☐
5. La valeur de `2*[[1,2]]` est [1,2,1,2].	☐	☐
6. La boucle if ….. est une boucle conditionnelle	☐	☐
7. L'instruction abs(a) renvoie la valeur absolue de a	☐	☐
8. Avec `for i in range(10)`, la variable i prend 9 valeurs puisque la dernière est 9.	☐	☐
9. Avec `for i in range(9,0,-3)`), les valeurs successives prises par la variable i sont 9,6,3	☐	☐
10. L'expression `str(list("ab"))` a pour valeur « ab »	☐	☐

EXERCICE 2 QCM

Pour chaque question, une seule réponse est correcte par les quatre proposées.
Vous avez le droit d'utiliser IDLE pour tester avant de répondre.

Question 1 : Le langage Python a été créé en :

- ☐ 1971
- ☐ 1991
- ☐ 2001
- ☐ 2011

Question 2: Parmi les propositions, laquelle n'est pas une expression ?

- ☐ a < b
- ☐ a != b
- ☐ a = b
- ☐ a >= b

Question 3: On considère les instructions suivantes exécutées dans l'ordre : a=8,b=5,a==b+1,b=b+1,a==b+1,b=b+1,print(a==b+1). Quel est le résultat affiché ?

- ☐ 8
- ☐ Aucun, une erreur est signalée
- ☐ False
- ☐ True

Question 4: On considère les deux instruction a=a+b et b=a-b exécutées dans l'ordre. Quelle affirmation est exacte ?

- ☐ Si les valeurs initiales de a et b sont respectivement « bon » et « jour » alors le programme est interrompu par une erreur.
- ☐ Si les valeurs initiales de a et b sont respectivement « bon » et « jour » alors les valeurs finales sont « bonjour » et « bon ».
- ☐ Si les valeurs initiales de a et b sont respectivement 5 et 2 alors les valeurs finales sont 7 et 3.
- ☐ Si les valeurs initiales de a et b sont respectivement 5 et 2 alors les valeurs finales sont 7 et 2.

Question 5: Combien de fois la fonction pint est-elle appelée dans le code en Python qui suit ?

```
n=4
for i in range(2,4):
    print(i)
```

- ☐ Jamais
- ☐ 1 fois
- ☐ 2 fois
- ☐ 3 fois

Question 6: Voici un code en Python :

```
x = 1
for i in range(4):
    x = x + i
```

Quel est la valeur finale de x ?

- ☐ 6
- ☐ 7
- ☐ 10
- ☐ 11

Question 7: Après le code Python qui suit, quelles sont les valeurs finales de x et de y ?

```
x = 4
while x > 0 :
    y = 0
    while y < x :
        y = y + 1
        x = x - 1
```

- ☐ La valeur finale de x est -1 et celle de y est 0
- ☐ La valeur finale de x est 0 et celle de y est 0
- ☐ La valeur finale de x est 0 et celle de y est 1
- ☐ La boucle externe est une boucle infinie, le programme ne termine pas

EXERCICE 3 LIRE DU CODE

Question 1: Voici le script

```
x = 1
n = 5
while n > 1:
    x = x * n
    n = n - 1

print(x)
```

Que va afficher l'instruction : print(x) ?

Question 2: Voici le script

```
x = 0
for i in range(2):
    x = x + i
    for j in range(3):
        x = x + j
print(x)
```

Que va afficher l'instruction : print(x) ?

Séquence 1
Programmation en Python – partie 1
CORRECTION DES EXERCICES

EXERCICE 1 VRAI / FAUX - Correction

	VRAI	FAUX
1. Voici les instructions suivantes : `x=3, y=5, x=y, y=x` La valeur de x est 5 et la valeur de y est 3.	☐	X
2. Voici les instructions suivantes : `x=3, y=5, y==x, x=y` La valeur de x est 5 et la valeur de y est 5.	X	☐
3. La valeur de `len("au revoir")` est 9.	X	☐
4. La valeur de `"toutou"*2` est « toutoutoutou ».	X	☐
5. La valeur de `2*[[1,2]]` est [1,2,1,2].	☐	X
6. La boucle if ….. est une boucle conditionnelle	☐	X
7. L'instruction abs(a) renvoie la valeur absolue de a	☐	X
8. Avec `for i in range(10)`, la variable i prend 9 valeurs puisque la dernière est 9.	☐	X
9. Avec `for i in range(9,0,-3) )`, les valeurs successives prises par la variable i sont 9,6,3	X	☐
10. L'expression `str(list("ab"))` a pour valeur « ab »	☐	X

EXERCICE 2 QCM - Correction

Pour chaque question, une seule réponse est correcte par les quatre proposées.
Vous avez le droit d'utiliser IDLE pour tester avant de répondre.

Question 1 : Le langage Python a été créé en :

- 1971
- **1991**
- 2001
- 2011

Question 2: Parmi les propositions, laquelle n'est pas une expression ?

- a < b
- a != b
- **a = b.**
  Les proposition 1,2,4 sont des expressions à valeurs booléennes. La proposition 3 est une instruction d'affectation.
- a >= b

Question 3: On considère les instructions suivantes exécutées dans l'ordre : a=8,b=5,a==b+1,b=b+1,a==b+1,b=b+1,print(a==b+1). Quel est le résultat affiché ?

- 8
- Aucun, une erreur est signalée
- False
- **True.** ***Il ne faut pas confondre = et ==. La valeur finale de a est 8 et celle de b est 7.***

Question 4: On considère les deux instruction a=a+b et b=a-b exécutées dans l'ordre. Quelle affirmation est exacte ?

- **Si les valeurs initiales de a et b sont respectivement « bon » et « jour » alors le programme est interrompu par une erreur.**
  L'opérateur – ne peut pas être utilisé avec le type str.
- Si les valeurs initiales de a et b sont respectivement « bon » et « jour » alors les valeurs finales sont « bonjour » et « bon ».
- Si les valeurs initiales de a et b sont respectivement 5 et 2 alors les valeurs finales sont 7 et 3.
- Si les valeurs initiales de a et b sont respectivement 5 et 2 alors les valeurs finales sont 7 et 2.

Question 5: Combien de fois la fonction pint est-elle appelée dans le code en Python qui suit ?

```
n=4
for i in range(2,4):
    print(i)
```

- Jamais
- 1 fois
- **2 fois**
  La variable i prend successivement les valeurs 2 et 3. Il y a donc deux passages dans la boucle.
- 3 fois

Question 6: Voici un code en Python :

```
x = 1
for i in range(4):
    x = x + i
```

Quel est la valeur finale de x ?

- ☐ 6
- ☐ 7

  La variable i prend successivement les valeurs 0,1,2 et 3, donc x prend les valeurs 1,2,4 et 7.
- ☐ 10
- ☐ 11

Question 7: Après le code Python qui suit, quelles sont les valeurs finales de x et de y ?

```
x = 4
while x > 0 :
    y = 0
    while y < x :
        y = y + 1
        x = x - 1
```

- ☐ La valeur finale de x est -1 et celle de y est 0
- ☐ La valeur finale de x est 0 et celle de y est 0
- ☐ **La valeur finale de x est 0 et celle de y est 1**

  Les valeurs de x sont strictement décroissantes et la valeur de y est remise à 0 dès qu'elle n'est plus strictement inférieur à celle de x. Au dernier passage dans la boucle interne, y vaut 0 et x vaut 1 : y prend la valeur de 1 et x la valeur 0 ; On sort de la boucle interne puis de la boucle externe.

  On a une boucle infinie si on remplace y=0 par y=1
- ☐ La boucle externe est une boucle infinie, le programme ne termine pas

EXERCICE 3 LIRE DU CODE - Correction

Question 1: Voici le script

```
x = 1
n = 5
while n > 1:
    x = x * n
    n = n - 1

print(x)
```

Que va afficher l'instruction : print(x) ?

La variable n a pour valeur 5 donc on entre dans la boucle. Il y a 4 passages successifs.

1er : x prend la valeur 5 et n la valeur 4

2e : x prend la valeur 20 et n la valeur 3

3e : x prend la valeur 60 et n la valeur 2

4e : x prend la valeur 120 et n la valeur 1.

La valeur de n est de 1 donc on sort de la boucle.

x a pour valeur finale : 120.

Question 2: Voici le script

```
x = 0
for i in range(2):
    x = x + i
    for j in range(3):
```

```
        x = x + j
print(x)
```

Que va afficher l'instruction : print(x) ?

Boucle externe : la variable i prend la valeur 0 et la variable x prend la valeur 0

Boucle interne : la variable j prend successivement les valeurs 0,1,2 donc la variable x prend successivement les valeurs 0,1,3

Boucle externe : la variable i prend la valeur 1 et la variable x prend la valeur 4.

Boucle interne : la variable j prend successivement les valeurs 0,1,2 donc la variable x prend successivement les valeurs 4,5,7

La variable x a donc pour valeur finale 7.

Extrait de cours

DEVOIR N°1 Programmation en Python partie 1

Répondez aux questions du QCM sans utiliser d'ordinateur.

Exercice 1 | QCM

1. **Python est un langage ….**
   a. Interprété
   b. Machine
   c. Compilé
   d. Binaire

2. **En Python, laquelle de ces fonctions converties une variable de type `string` en `float` :**
   a. str(x)
   b. float(x)
   c. x.float()
   d. inToFloat(x)

3. **Soit le code suivant :**

```
a="Bon"
b="jour"
print(b+a)
```

Qu'affiche le script ?
   a. jourBon
   b. Bonjour
   c. b+a

4. **Soit le code suivant :**

```
toto = 'Bonsoir'
print(toto[3:7])
```

Qu'affiche le script ?
   a. toto[3:7]
   b. soir
   c. nsoir

5. **Soit le code suivant :**

```
a = 2,5
b = 1,8
print(a*b)
```

Qu'affiche le script ?
   a. 4.5
   b. 4,5
   c. Erreur

6. **Soit le code suivant :**

```
a = '10'
b = '2'
c = a + b
print(c)
```

Qu'affiche le script ?
   a. 12
   b. 102
   c. 210

7. Soit le code suivant :

```
a=7
b=12
if a>5:
    b=b-4
if b>=10:
    b=b+1
```

Que vaut la valeur finale de la variable b ?

a. 8
b. 9
c. 12
d. 13

8. Soit le code suivant :

```
a=3
b=6
if a>5 or b!=3:
    b=4
else:
    b=2
```

Que vaut la valeur finale de la variable b ?

a. Erreur
b. 2
c. 4
d. 6

9. Soit le code suivant :

```
a=2
b=5
if a>8:
    b=10
elif a>6:
    b=3
```

Que vaut la valeur finale de la variable b ?

a. Erreur
b. 3
c. 5
d. 10

10. Soit le code suivant :

```
a=2
b=0
if a<0:
    b=1
elif a>0 and a<5:
    b=2
else:
    b=3
```

Que vaut la valeur finale de la variable b ?

a. 0
b. 1
c. 2
d. 3

11. Soit le code suivant :

```
n = 0
while n<15 :
    n = n + 2
print(n)
```

Qu'affiche le script ?

a. 14
b. 15
c. 16
d. 17

12. Soit le code suivant :

```
n = 10
while n>=11 :
    n = n + 2
print(n)
```

Qu'affiche le script ?

a. 10
b. 11
c. 12
d. 13

13. Soit le code suivant :

```
n = 0
for i in range(5) :
    n = n + 1
print(n)
```

Qu'affiche le script ?

a. 4
b. 5
c. 6
d. 7

14. Soit le code suivant :

```
n = 0
for i in range(5) :
    n = n + 1
print(i)
```

Qu'affiche le script ?

a. 4
b. 5
c. 6
d. 7

15. Soit le code suivant :

```
resultat = ""
for c in "Bonsoir" :
    resultat = resultat + c
print(resultat)
```

Qu'affiche le script ?

a. Bonsoir
b. riosnoB
c. BonsoirBonsoirBonsoirBonsoirBonsoirBonsoirBonsoir

Exercice 2 | Écrire du code en Python

Écrire un programme en Python, qui permet de calculer l'aire d'un triangle lorsque l'utilisateur saisi :

- La hauteur
- La base