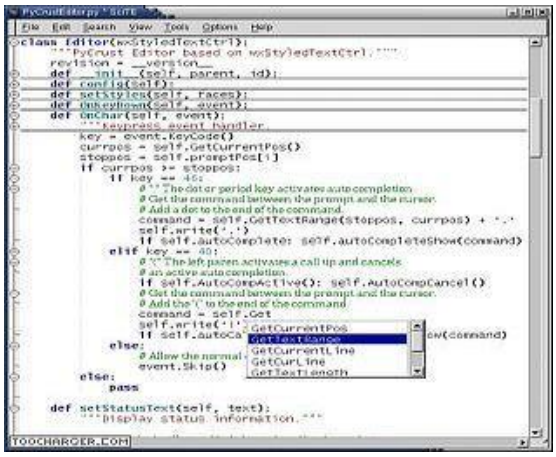
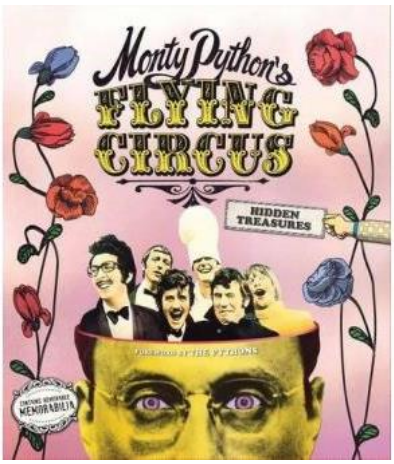
DEBUTER EN PROGRAMMATION PYTHON : *CORRIGE* INGREDIENTS DE BASE



+



=

- *Matériel : Ordinateur, accès Internet.*
- *Logiciels et sites internet : Thonny, pythontutor.com.*
- *Pré-requis pour prendre un bon départ :*

	☹	😐	☺	☺☺
Différencier une grandeur et sa valeur ou, ce qui revient au même, une fonction et la valeur d'une fonction.				
Manipulations informatiques basiques : ouvrir, copier, enregistrer un fichier ; aller et trouver sur Internet etc.				
Télécharger et installer un programme.				

NOM et prénom : …… *Première spécialité NSI*

I. PREPARATIFS.

A. Comment utiliser ce livret :

Contrairement au titre un peu présomptueux, ce livret, seul, ne pourra pas vous donner la science pythonesque. Ce livret doit être utilisé en conjonction avec la plateforme d'apprentissage de France IOI sur le Web.

La pédagogie choisie est donc celle du self-learning (apprentissage par soi-même) inversé : la pratique d'abord sur France IOI puis la théorie ensuite avec ce livret.

Ce livret est donc à considérer comme un post-cours qui permet de remettre les connaissances à plat, d'éclairer sur la syntaxe (l'écriture) du langage ou sur un concept et donc ainsi de faire le point.

B. Le site internet France IOI :

Ce super site labellisé Education Nationale, est le site d'entraînement et de sélection de l'équipe de France aux Olympiades Internationales d'Informatique (IOI).

Cet entraînement est ouvert à tous, il suffit de s'inscrire :

- En haut à gauche s'inscrire (Se connecter).
- **Puis dans son compte, remplir son VRAI NOM, son VRAI Prénom.**
- Rejoindre dans la foulée le groupe **lasource2020**.

Ça y est, vous êtes fin prêt pour commencer les entraînements (partie Progresser / Cours et problèmes). Cette année, nous ferons sur France IOI au moins les niveaux 1 et 2.

Pour programmer, vous aurez besoin d'**un environnement de développement à installer sur votre propre machine.**

C. L'environnement de développement :

Pour écrire, éditer et enregistrer des programmes, il faut travailler dans un environnement prévu à cet effet :

l'Environnement de Développement Intégré (IDE en anglais).

L'environnement de développement fourni par le site officiel de Python est trop basique : par exemple, il n'y a même pas de numérotation automatique des lignes du programme !

Heureusement, il existe beaucoup d'autres environnements bien plus performants fournis par d'autres éditeurs : VS Code de Microsoft, Atom de Github, Repl.it en ligne etc. et Thonny.

Thonny est un environnement de développement créé en 2015 par Aivar Annamaa (Université de Tartu - Estonie). Son interface simple a été pensée pour les débutants en Python.

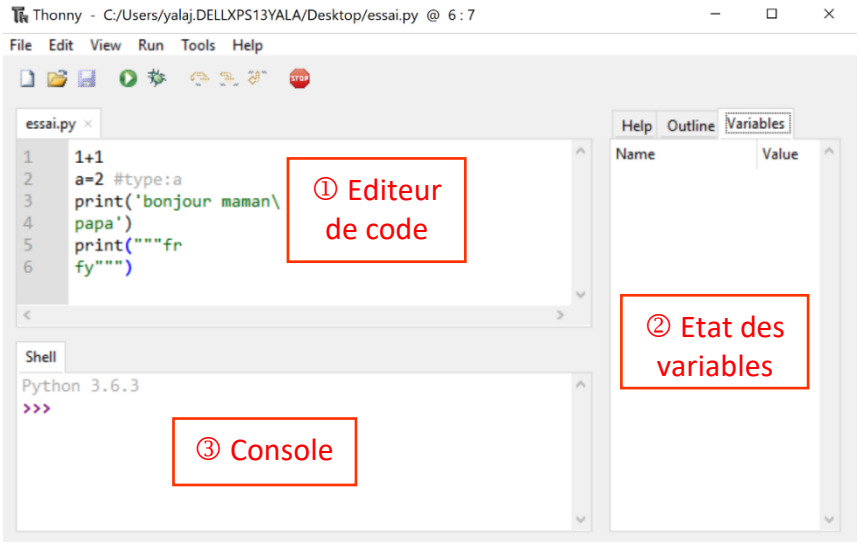
1. Installation de Thonny :

Aller sur le site de Thonny.org et télécharger la dernière version puis l'installer.

2. Options de configuration à activer :

- Menu Tools / Options / Général : Langage → Français.
- Menu **Outils / Options / Editeur : tout cocher.**
- **Menu View : cocher : Aide – Assistant – Console – Variables.**

  Vous devriez obtenir cela →



3. Les 3 parties de l'interface de Thonny :

1. **L'éditeur de code** : c'est là qu'on écrit, modifie un programme.

   Cet éditeur bénéficie de la coloration syntaxique et de la complétion automatique de code (touche tab).
2. **La colonne d'états des variables** : là on peut voir la valeur de chaque variable.
3. • **La console (Shell en anglais) est le terminal d'entrées-sorties**. C'est là que se font :

   les entrées de données (correspondant aux input ( ) du programme) par le clavier.

   les affichages-sorties à l'écran (correspondant aux print( ) du programme).

   • **<u>La console est aussi un testeur d'instructions très pratique.</u>**

   Elle permet de tester une ligne de code à la fois (pas plusieurs). Très utile par exemple pour tester une fonction, vérifier une syntaxe etc.

   <u>Exemples :</u> Taper dans la console et écrire le résultat :

   2 + 3 / 3 → *5* type(5) → *<class 'int'>* 4 = = 1 + 2 → *False*

   print('Hello') → *Hello* print(Hello) → *NameError: name 'Hello' is not defined*

4. Pour installer des modules externes lorsqu'on en aura besoin :

- Menu Outils / Gérer les paquets (Manage packages).
- Entrer le nom du module désiré (Pygame par exemple) puis Rechercher sur Pypi.
- Install.

Ai-je tout compris ? Préparatifs.	☹	😐	☺	☺☺
Comment utiliser ce cours ?				
Inscription sur France IOI et avoir rejoint le bon groupe.				
Qu'est-ce que France IOI ?				
Qu'est-ce qu'un IDE ?				
Installer et configurer Thonny.				
Les différentes parties de l'interface de Thonny.				
La console (le shell).				

D. Avant de commencer :

➢ Je rappelle que ce post-cours est à utiliser après s'être exercé sur le site de France IOI.

Voici à peu près la correspondance des chapitres de ce livret avec le niveau 1 de France IOI :

Exercices France IOI	*Chapitres de ce livret*
Niveau 1 Chapitre 3.	Chapitre IV-V-VI : Variables, Types, Expressions
Niveau 1 Chapitres 1-3.	Chapitre VII : Instructions.
Niveau 1 Chapitres 1-3-5.	Chapitre VIII : Instructions de base.

➢ Dans la suite du cours, pensez au moindre doute à **vérifier un résultat, une syntaxe etc. avec la console de Thonny** ou sur le site pythontutor.com (moins complet mais pratique).

➢ Avant de vraiment pythoner, il nous reste à aborder quelques généralités sur les langages de programmation.

II. LANGAGES DE PROGRAMMATION.

A. Dualité Algorithme – Programme :

- Un programme est un texte qui permet de faire exécuter un *algorithme*[1] à une machine.

Ce texte est écrit dans un langage particulier appelé « langage de programmation ».

- Par analogie langagière, l'algorithme est l'histoire racontée par ce texte et le langage de programmation est la langue dans laquelle est écrite cette histoire.

➢ Produire un bon gros programme est donc la rencontre d'une bonne histoire (un algorithme intéressant, efficace etc.) avec un bon style (une bonne maîtrise de la programmation).

Trouver une bonne histoire relève de l'Algorithmique (voir cours sur les Algorithmes). Cette compétence passe forcément par l'étude des grands classiques (algorithmes de recherche, de tri etc.).

Avoir du style relève de la programmation. Cette capacité passe par l'étude du langage de programmation.

➢ Apprendre un langage de programmation, c'est d'abord apprendre le vocabulaire et la grammaire attachés à ce langage, ce qui permettra d'écrire de petits programmes ou scripts (de petites histoires, de petites mélodies sans prétention) puis on avance vers des textes plus ambitieux.

B. Diversité des langages de programmation :

➢ De la même manière qu'il existe environ 7 000 langues vivantes dans le Monde dont environ 200 écrites[2], il existe près d'1 millier de langages de programmation : *Rust, Go, Python, Javascript, C, etc.*[3]

Le classement IEEE Spectrum donne les langages informatiques les plus utilisés dans le Monde. Lesquels ?

➢ Les langages de programmation sont en général classés selon 2 critères (voir sur Internet) :

- leur(s) paradigme(s) de programmation (leur approche de la programmation) :
  Paradigme impératif, paradigme orienté objet, paradigme fonctionnel etc.
- leur typage (la façon dont est indiqué le type des variables) :
  Typage explicite ou implicite, typage statique ou dynamique, typage fort ou faible.

Exemples : Pour les langages de programmation suivants, indiquer leur paradigme et leur typage.

Langages	*Paradigme(s) ou style(s) de programmation*	*Typage des variables*
Python	*impératif, orienté objet, fonctionnel*	*implicite – dynamique – fort*
Langage C	*impératif*	*explicite – statique – faible*
Haskell	*fonctionnel*	*implicite – statique – fort*

➢ D'autres critères encore permettent de différencier les langages de programmation :

- comment sont exécutées les instructions ? : langages interprétés (Python), langages compilés (C).
- la plus ou moins proximité avec le langage binaire-machine qui est le langage de plus bas niveau.

---

[1] Rappel : « Un algorithme est une suite finie et non ambigüe d'instructions et/ou d'opérations permettant d'effectuer une tâche ou résoudre un problème en un temps fini. Le mot algorithme vient du nom latinisé du mathématicien perse Al-Khawarizmi (IX siècle ap. J.C.).
Le domaine qui étudie les algorithmes est appelé l'Algorithmique. » Wikipédia.

[2] http://www.axl.cefan.ulaval.ca/Langues/1div_recens.htm.

[3] https://fr.wikipedia.org/wiki/Liste_de_langages_de_programmation.

➢ Il n'est pas nécessaire d'apprendre un maximum de langages : en effet, tous les langages ayant le même paradigme reposent plus ou moins sur les mêmes concepts de base.

Le paradigme le plus simple est le paradigme « impératif » : par enchaînement d'instructions.

C'est par ce type d'approche que tout le monde apprend la programmation et ce sera le cas pour nous cette année de Première. En Terminale NSI sera abordé le paradigme « orienté objet ».

Ai-je tout compris ? Langages de programmation.	☹	😐	☺	☺☺
Qu'est-ce qu'un programme ?				
Qu'est-ce que le paradigme d'un langage de programmation ?				
Citer quelques paradigmes de programmation.				
Qu'est-ce que le typage d'un langage de programmation ?				
Citer une sorte de typage.				
Citer d'autres critères qui permettent de différencier les langages de program.				

III. LE LANGAGE PYTHON : GENERALITES.

Ce cours concerne le très populaire **langage de programmation libre et multiplateformes Python dans sa version≥ 3.7.**

D'après le B] page précédente, on sait que Python est un langage :

- multi-paradigmes : *impératif, orienté objet et fonctionnel.*
- à typage *implicite, dynamique fort.*

A. Python dans le Monde :

➢ Ce langage créé en 1991 par le mathématicien hollandais Guido Van Rossuma été baptisé ainsi en l'honneur du célèbre groupe de comiques anglais The Monthy Python's Flying Circus.

➢ Python est un langage plutôt simple pédagogiquement (écriture-syntaxe simple, peu de mots clé à retenir au début).

Ainsi, en 2014, Python avait dépassé Java comme langage d'initiation à la programmation des étudiants américains : 8 des 10 cursus en informatique des meilleures universités américaines s'en servaient pour enseigner le codage. Et c'était aussi le cas de 27 des 39 grandes écoles d'Informatique des Etats Unis.

De même, Python est le langage qui a été choisi par l'Education Nationale pour l'apprentissage de la programmation au Lycée (langage par blocs Scratch au Collège).

➢ N'allez pas croire que Python est cantonné à l'apprentissage de la programmation !

Il est aussi utilisé dans des applications lourdes et sensibles : par exemple dans le secteur financier pour l'analyse des données. Python est aussi très présent dans le domaine de l'Intelligence Artificielle et du Big Data. C'est donc un langage professionnel très demandé par les employeurs au même titre que Java et C.

B. Les ingrédients fondamentaux des programmes :

Apprendre la programmation, ce n'est pas seulement apprendre à écrire un programme mais c'est aussi comprendre de quoi il est fait, comment il est fait et ce qu'il fait.

Voici un petit programme écrit en Python :

File Edit Format Run Options Window Help

```
a = 7
b = 4
print ("A vous de jouer.")
x = int(input())
y = int(input())
if x == a and y == b :
    print("Coulé")
else:
    if x == a or y == b :
        print ("en vue")
    else :
        print("Dans l'eau")
```

1. Que fait ce programme ? Quel nom pourrait-on lui donner ?

A la vue des phrases « Coulé », « En vue » et « Dans l'eau », on se doute bien qu'il s'agit du jeu de la bataille navale.

2. De quoi est composé ce programme ?

- *De données enregistrées dans des variables : a , b ← valeurs prédéfinies 7 , 4 ; x et y ← valeurs entrées par l'utilisateur*
- *Des instructions de contrôle : entrée d'infos (input()), sortie d'infos (print( )), test conditionnel if.*

Un programme est essentiellement constitué d'expressions ou structures de données et d'instructions ou structures de contrôles. Les expressions sont les objets, les sujets. Les instructions sont les actions, les verbes.

Le cours d'algorithmique nous apprendra qu'il y a 4 ingrédients de base quel que soit le programme :

- **1 structure de données : les variables (qui représenteront l'Information).**
- **3 structures de contrôle du flux d'instructions (qui permettront de traiter cette Information) :**
  - **les séquences d'instructions.**
  - **les tests conditionnels ou alternatives.**
  - **les boucles ou répétitions.**

Ai-je tout compris ? Python généralités.	☹	😐	☺	☺☺
Qu'est-ce que Python ?				
Quels sont le paradigme et le typage du langage Python ?				
Quels sont les 4 structures fondamentales des programmes ?				

IV. STRUCTURES DE DONNEES : LES VARIABLES.

Un programme a pour but d'effectuer une tâche ou de résoudre un problème. Forcément, certaines choses ont varié après exécution de ce programme car on est passé d'un état non traité à un état traité.

Ces choses qui varient s'appellent évidemment des ***variables.***

A. Qu'est-ce donc ?

Une variable est une donnée ponctuelle du programme stockée dans la mémoire de l'ordinateur.
Une variable a 2 caractéristiques : son nom et sa valeur.

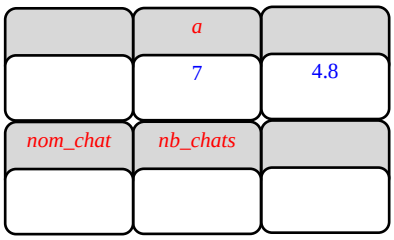
➢ Il faut imaginer la mémoire de l'ordinateur comme une grosse armoire avec plein de petites boîtes.

Certaines de ces boîtes ont un nom (une étiquette) : ce sont des variables, et elles peuvent contenir une valeur.

➢ En reprenant l'exemple p.7 de la bataille navale, lorsque l'instruction « a = 7 » est exécutée, une des petites boîtes prend pour nom « a » (c'est la déclaration de la variable) et la valeur « 7 » est mise dans cette boîte (c'est l'affectation de valeur à la variable).

La variable « a » est le conteneur et la valeur « 7 » est le contenu.

Illustration



B. Déclaration de variable :

Déclarer une variable, c'est en fait *nommer* cette variable.
Dans la machine, une boîte mémoire est réservée et le nom de la variable est attribué à cette boîte mémoire.

Sur ce point, le langage Python demande une moindre rigueur que d'autres langages.

Par exemple, en langage Pascal, il est obligatoire de déclarer (et typer) une variable au tout tout début du programme avant les instructions !

Alors qu'en Python, la déclaration d'une variable (et son typage) se fait dynamiquement à l'exécution du programme dès que cette variable apparait dans une ligne de code.

C. Règles de nommage d'une variable :

1. Le nom d'une variable : plusieurs obligations.

- ne peut pas être l'un des 29 + 3 mots réservés du langage Python : *if, for, while, print, with etc.*
- **ne peut jamais commencer par un chiffre.**
- est exclusivement composé de lettres Majuscules et/ou de minuscules accentuées ou pas, de chiffres.
  Seul symbole autorisé : souligné « _ » (underscore en anglais).
  Tous les autres caractères spéciaux sont interdits, en particulier l'espace blanc « » !
- est sensible à la casse : les minuscules sont différentes des MAJUSCULES. Exemple : Age ≠ age.

2. Le nom d'une variable doit être parlant !

IL FAUT CHOISIR DES NOMS LONGS ET PARLANTS POUR SES VARIABLES AFIN QUE LE PROGRAMME SOIT LISIBLE PAR N'IMPORTE QUI D'AUTRE QUE LE CONCEPTEUR.

Exemples : liste_notes_trim1 et non n ou ntri1 ou notes ; fichier_ouvert et non f ou ou fo ou fic ou fichier.

Astuce : Utiliser une ou plusieurs fois l'underscore « _ » dans le nom de la variable !

Le nommage des variables est l'une des règles les plus difficiles à suivre par les débutants qui ont la flemme d'écrire et mal habitués j'avoue par les Mathématiques qui utilisent souvent des lettres uniques pour désigner des inconnues ou des variables.

3. Application :

Entourer : en rouge les noms de variable entraînant une erreur d'exécution.

en bleu les noms de variables n'entraînant pas d'erreurs d'exécution mais mal choisis.

valeurs A' beau_pere nb beauperes nb_belle-meres belle fille nb_beau_fils. Tableau _zarbi int Temp_Corps temp-peau TC nan_mais_allo_quoi False_True 1ereS n1 var

Ai-je tout compris ? Variables.	☹	😐	☺	☺☺
Qu'est-ce qu'une variable ?				
Que se passe-t-il dans la machine lorsqu'on déclare une variable ?				
Règles de nommage des variables ?				

Voyons maintenant quels types de données peuvent contenir ces variables.

V. TYPES DE DONNEES DE BASE.

➢ On n'a manipulé jusqu'à maintenant que des nombres. Toutes les données à manipuler ne sont heureusement pas que des nombres !

Il existe en Python (comme dans tout autre langage de programmation) d'autres types de données, ce qui apporte de la souplesse dans la manipulation de ces données.

La liste complète est ici : https://fr.wikiversity.org/wiki/Python/Les_types_de_base.

➢ Ci-dessous les 4 types de données basiques incontournables :

Type	*Signification (anglais)*	*Signification (français)*	*Description*
int	Integer	Entier relatif	Entier relatif compris entre -2^{n-1} – 1 et 2^{n-1} pour un codage sur n bits(voir cours codage binaire).
float	Floating point number	Nombre à virgule flottante	Représente un nombre réel **approximativement.** **S'écrit avec un point** (ex : 2.68) et non une virgule !
str	Character string	Chaîne de caractères	Chaîne de caractères : un mot, une phrase… S'écrit entre ' ' ou entre " " ou entre """ """.
bool	Boolean	Booléen	Une variable booléenne est (⚠ **M**ajuscule) : soit égale à **T**rue (vrai) ; soit égale à **F**alse (faux).

A. Nombres :

Il existe 3 types de nombres en Python : int, float et complex.

Ils seront vus plus en détail dans le cours sur la représentation des nombres en langage binaire.

1. Type int : nombres entiers.

2. Type float : nombres à virgule flottante (pseudo nombres réels).

3. Type complex : nombres complexes.

B. Chaînes de caractères :

1. Type string character (str) :

Tout caractère utilisé dans Python provient de la table Unicode (cours Numérisation des textes p.6) qui regroupe quasiment tous les caractères utilisés dans le Monde.

Dans cette table, chaque caractère correspond à un unique numéro appelé point de code.

2. Chaîne de caractères : définition.

Une chaîne de caractères est une « liste » de caractères tous de type string (str) collés les uns aux autres.

Nous reviendrons au cours4 Listes sur cette notion de chaîne de caractères vue en tant que liste.

3. Trois façons d'afficher une chaîne de caractères :

Pour cela, nous allons utiliser la **fonction print( ).** Nous la reverrons en détail p.21 : instructions de sortie.

Manière	*Exemples : print('chaîne de caractères') → résultat à l'écran*	*Commentaires*
Entre 'apostrophes' droits (single quotes en anglais)	print('Mes parents chéris,') → Mes parents chéris,	Impossible d'aller à la ligne dans print( ).
Entre "guillemets" droits (double quotes en anglais)	print("je vous aime.") → je vous aime.	Impossible d'aller à la ligne dans print( ).
Entre """triple guillemets""" droits (triple double quotes en anglais)	print("""Votre enfant qui pense à vous.""") → Votre enfant qui pense à vous.	• Tient compte des retours à la ligne. • Marche aussi avec triple simple quotes.

4. Apostrophes ou guillemets à l'intérieur d'une chaîne de caractères :

Exécuter print('Chérie je t'aime.'). Que s'affiche-t-il ? *SyntaxError: invalid syntax*

Si une chaîne de caractères contient des guillemets ou des apostrophes, 4 solutions :

Manière	*Exemples : print(Chaîne de caractères)→ résultat à l'écran*
Délimiter par des apostrophes une chaîne contenant des guillemets (mais pas d'apostrophes).	print('Elle lui a dit : "Nan mais allo koi !". Lol.') → Elle lui a dit : "Nan mais allo koi !". Lol.
Délimiter par des guillemets une chaîne contenant des apostrophes (mais pas de guillemets).	print("C'est celui qui dit qui y est !") → C'est celui qui dit qui y est !
Délimiter par des triples guillemets une chaîne contenant des apostrophes et/ou guillemets.	print("""Ce à quoi elle a rétorqué : « T'es qu'un pov type, sale caractère ! »""") →Ce à quoi elle a rétorqué : « T'es qu'un pov type, sale caractère ! »
Précéder chaque apostrophe ou chaque guillemet par le caractère échappatoire antislash \ .	print('\' Est-ce que tu m\'aimes encore ? \' ') →' Est-ce que tu m'aimes encore ? '

5. Quelques rôles du caractère échappatoire antislash \ :

Syntaxe	*Exemples :* *print(Chaîne de caractères)→ résultat à l'écran*	*Commentaires*
\' **\"** ****	print("Momo dit : **\"**Tu tires **** tu pointes ?**\"** ") → Momo dit : "Tu tires \ tu pointes ?"	Permet d'écrire les caractères apostrophe, guillemet, antislash lui-même dans une chaîne de caractères sans provoquer de SyntaxError.
\n	print("Momo dit : **\n** \"Tu tires \\ tu pointes ?\" ") → Momo dit : "Tu tires \ tu pointes ?"	Insère un retour à la ligne là où \n est placé.
\t	print("Momo dit : **\t** \"Tu tires \\ tu pointes ?\" ") → Momo dit : « Tu tires \ tu pointes ? »	Insère une tabulation là où \t est placé.

Il existe encore beaucoup de rôles pour l'antislash \ : voir https://python-course.eu/python3_variables.php.

Citons celui-ci très pratique :

\ placé en fin de ligne	print("Je suis venu te dire\ que je m'en vais. ") → Je suis venu te dire que je m'en vais.	Permet de revenir à la ligne dans l'écriture de n'importe quelle (longue) ligne d'instruction. Sans effet sur l'exécution du programme.

6. Application :

Ces phrases vont-elles correctement s'afficher à l'écran ? Vérifier à la console si besoin. Non.

print (' j'aime mon prof. ') print (" j'aime mon prof.) print (' j\'aime mon prof. ')

print (" j\'aime mon prof.") print ("j'aime mon prof ")

C. Booléens :

Un booléen est un objet qui ne peut prendre que 2 valeurs : soit True (*vrai*), soit False (*faux*). En général, un booléen est une expression posant une question (? sous-entendu). Exemples : « 1 = = 5 – 4 », « 5 > k ».

Evidemment, il existe d'autres types de données plus complexes (list : liste, file : fichier etc.).

On pourra même créer les propres types complexes dont on aura besoin (paradigme orienté objet).

Ai-je tout compris ? Types de données.	☹	😐	☺	☺☺
Citer 4 types importants de variables.				
3 façons de délimiter une chaîne de caractères.				
Afficher du texte contenant des apostrophes, guillemets ou des antislashs.				
Utiliser l'antislash dans des chaînes de caractères ou en fin de ligne d'instruction.				
Qu'est-ce qu'une variable booléenne ?				

Maintenant, comment agir entre ces valeurs/variables ? C'est l'objet des opérateurs.

VI. OPERATEURS, DELIMITEURS ET EXPRESSIONS.

Un opérateur permet d'agir entre 2 valeurs et/ou variables.

En combinant les opérateurs, on peut créer des expressions plus ou moins complexes selon les besoins. Il faudra alors faire attention à l'ordre d'exécution des opérateurs.

Les délimiteurs seront utiles pour changer ou bien faire ressortir les priorités.

A. Opérateurs :

Les opérateurs correspondent à certaines opérations mathématiques et bien plus.

1. Définitions :

1. **Un *opérateur* est un symbole (ou un mot réservé) utilisé pour effectuer une opération entre 2 *opérandes*.**
2. Un **opérande** (masculin !) est soit une valeur, soit une variable, soit une expression.
3. Un **couple de délimiteurs** sont 2 symboles qui permettent d'encapsuler une expression.
4. Une **expression** est une suite valide d'opérateurs, de délimiteurs et d'opérandes.

Exemple : Soit l'expression (nbsmacks+1)*2

1 couple de délimiteurs « ( ) » ; 2 opérateurs : « + » et « * » ; 3 opérandes : « nbsmacks », « 1 » et « 2 ».

2. Opérateurs mathématiques :

➢ Compléter les exemples du tableau suivant puis vérifier si besoin avec la console.

Symbole	*Nom*	*Type des 2 opérandes*	*Exemples à vérifier : expression → résultat*	*Commentaires*
**	Puissance	Int et/ou Float	4**2 → *16* 4**2.0 → *16.0* Ecrire nb^a : *nb**a*	Idem aux Maths. On peut aussi utiliser pow(. , .). Ex : pow (2 , 3) == 2**3.
*	Multiplication	Int et/ou Float	2 * 5.0 → 10.0	Idem aux Maths.
		Int et Caractères (str)	3 * "ab" → "ababab" "Pi" * 2→ *'PiPi'*	Répétition et accolement de la séquence de caractères.
/	Division décimale	Int et/ou Float	2 / 4 → 0.5 8 / 2 → 4.0 7 / 7 → *1.0*	Renvoie *toujours* un résultat de type réel (float).
//	Quotient entier	Int et/ou Float	5 // 3→1 5.0 // 3 → 1.0 2 // 3 → *0* 4.1 // 3.1 → *1.0*	Renvoie **la *partie entière* du quotient** (entier int quand tous les opérandes sont int ; sinon entier float).

Symbole	*Nom*	*Type des 2 opérandes*	*Exemples à vérifier : expression → résultat*	*Commentaires*
%	Reste (ou Modulo)	Int et/ou Float	5 % 3 → 2 5.1 % 3 → 2.1 2 % 3 → *2* 4.1 % 2.1 → *1.99etc6*	• Renvoie le reste (**pas forcément entier !**) de la division pseudo-euclidienne (entier int quand tous les opérandes sont int ; sinon réel float). • ***Très pratique pour faire une action toutes les n fois en utilisant l'expression « compteur % n ».***
+	Addition	Int et/ou Float	2 + 3 → 5 2 + 3.0 → *5.0*	Idem aux Maths.
		Str et Str	"A" + "ne" → *'Ane'*	Accolement ou **Concaténation.**
-	Soustraction	Int et/ou Float	2 - 3 → -1 2 - 3.0 → *-1.0*	Idem aux Maths.

➢ Remarques :

- On voit que les opérateurs + et * agissent différemment selon qu'on ait affaire aux types Nombres ou au type Caractères : on parle alors de **surcharge d'opérateur**.
- Tous ces opérateurs mathématiques marchent aussi avec des booléens considérés alors comme des entiers int : True valant évidemment l'entier *1* et False valant évidemment l'entier *0*.
- **Cela sera pratique pour par exemple évaluer la véracité de plusieurs conditions en même temps : il suffira d'additionner les conditions et voir combien on trouve.**
- Et la racine carrée ? Ce n'est pas un opérateur qui assure ce calcul mais une fonction présente dans le module math (à importer d'abord avec l'instruction « from math import * ») : la fonction **sqrt ( )**.

3. Opérateurs logiques ou opérateurs booléens :

Les expressions avec un opérateur logique renvoient toutes un booléen : *soit True soit False.*

Mot réservé	*Nom*	*Type des opérandes*	*Exemples à vérifier à la console : expression → résultat*	*Commentaires*
not	NON logique	1 seul opérande booléenne	not (1 > 3) → True not (2 < 9) → *False*	Renvoie l'opposé booléen (la négation).
and	ET logique	2 opérandes booléennes	(1 > 0) and (1 > 2) → *False* (1 > 0) and (1 < 2) → *True* (1 < 0) and (1 > 2) → *False* (1 < 0) and (1 < 2) → *False*	• Renvoie True seulement si les 2 opérandes sont vrais (V et V). • Renvoie False dans tous les autres cas (V et F, F et V, F et F).

Mot réservé	Nom	Type des opérandes	Exemples à vérifier à la console : expression → résultat	Commentaires
or	OU logique	2 opérandes booléennes	(1 > 0) or (1 > 2) → *True* (1 > 0) or (1 < 2) → *True* (1 < 0) or (1 > 2) → *False* (1 < 0) or (1 < 2) → *True*	• Renvoie False seulement si les 2 opérandes sont faux (F or F). • Renvoie True dans les 3 autres cas (V or F, F or V, V or V).

4. Opérateurs de comparaison :

➢ **Les expressions avec l'un des 8 opérateurs de comparaison posent toutes une question (point d'interrogation « ? » sous-entendu) dont la réponse est un booléen : soit True, soit False.**

Les 2 opérandes doivent être en même temps du « même » type et ce type doit posséder une relation d'ordre : c'est le cas pour les types nombres, caractères (grâce à leurs valeurs unicode), séquences etc.

Symbole (ou mot réservé)	Question posée	Type des 2 opérandes	Exemples à vérifier : expression → résultat	Commentaires
= =	Egalité de valeur ?	« même » type	3 = = 1 + 2 → *True* 1.0 = = 1 → *True* "1" = = 1 → *False*	Compare 2 objets de « même » type pour voir s'ils ont la même valeur.
!=	Différents ?	« même » type	3 != 1 + 2 → *False* 1.0 != 1 → *False* "1" != 1 → *True*	Contraire de ==. A != B équivaut à not (A = = B).
< <= > > =	Inférieur ? Inférieur ou égal ? Supérieur ? Supérieur ou égal ?	« même » type	1 < 0 → *False* 'c'< = 'h'→ *True* 'h'>'B'→ *False* 'fdb'> = 'fde'→ *True*	• Ordre naturel des nombres. • D'après la table Unicode, on a : ordre naturel des lettres. MAJUSCULES < minuscules.
is is not	Identiques ? Non identiques ?	N'importe quels types	1.0 is 1 → *False* 1 + 2 is not 3 → *False*	• Retourne True si A et B sont exactement le même objet. • A is not B équivaut à not (A is B). • L'opérateur is est beaucoup plus restrictif que l'opérateur = =. • Ne génère jamais de TypeError.

➢ Remarque :

- Si les expressions comparées ne sont pas du « même » type, une TypeError est générée (sauf pour l'opérateur is).
- On peut enchaîner les opérateurs de comparaison. Ex : **a < b < c équivaut alors à (a < b) and (b < c).**

5. Quelques autres opérateurs :

- Opérateur d'appartenance :

Mot réservé	*Nom*	*Type de l'opérande*	*Exemples à vérifier à la console : expression → résultat*	*Commentaires*
in not in	Dans (pas dans) la séquence ?	Séquence (list, tuple, binaires, caractères)	'py' in 'papy' → True 1 in [0 , 'a' , 1.0] → True 'k' in ('ko', 1) → *False* 'k' not in ['k', 1] → *False*	• « k in Séquence » : renvoie True si la valeur de k est égale à la valeur de l'un des items de Séquence.

- Opérateurs binaires :

Il existe des opérateurs spécialisés dans le traitement des écritures binaires (écritures avec des 0 et des 1). Voir le cours sur le langage binaire V] p.16.

B. Priorité des opérateurs et délimiteurs dans les expressions :

➢ Dans une expression (une suite valide d'opérateurs et d'opérandes) les opérations se font rarement de gauche à droite mais, comme en Mathématiques, selon des règles de *priorité.*

Ordre décroissant de priorité des délimiteurs et opérateurs (tableau non exhaustif)

	Délimiteurs ou Opérateur	*Nom*
opérateurs maths	( )	Délimiteurs Parenthèses.
	**	Puissance.
	+k ; -k	Positif ; Négatif.
	* ; / ; // ; %	Multiplication ; Division classique ; Quotient entier ; Reste.
	+ ; –	Addition ; Soustraction.
opérateurs booléens	< ; <= ; > ; >= ; != ; = =	Comparaisons de valeur.
	is ; is not	Test d'identité.
	in ; not in	Test d'appartenance.
	not	NON booléen.
	and	ET booléen.
	or	OU booléen.

➢ Remarques :

- **Les parenthèses sont le couple privilégié de délimiteurs. Elles permettent facilement :**
  - **de changer l'ordre des priorités.**
  - **de lever des ambiguïtés de priorité**.
- Pour des opérateurs de même niveau de priorité, les opérations sont évaluées de gauche à droite.

➢ Application : Evaluer les expressions suivantes puis vérifier à la console.

-(-5)**2 != 15 + 20 / 2 → *True* (1 < 3) or [(2 = = 1 + 2) and (1 > 0 + 2)] → *V or [F and F] → V or F → V*

[not (1 > 3)] and ('ta' in 'TaTa') → *V and F →F* ('Ab' > 'AC' > 'aC') or ('ac' != 'AB') → *F or V → V*

Ai-je tout compris ? Opérateurs, expressions.	☹	😐	☺	☺☺
Définitions d'un opérateur, d'un opérande, d'une expression.				
Opérateurs mathématiques.				
Opérateurs logiques.				
Opérateurs de comparaison.				
Evaluer une expression en respectant les priorités des opérateurs.				

VII. STRUCTURES DE CONTROLE DE BASE : INSTRUCTIONS.

A. Définitions : Instruction ; Séquence d'instructions.

- Une **instruction** est un ordre unitaire qui doit être exécuté par la machine.
- Chaque instruction se présente dans le programme sous la forme d'une ligne écrite appelée **ligne de code**. Chaque instruction s'écrit comme une combinaison d'expressions, d'opérateurs, de fonctions etc.
- Une suite de plusieurs instructions (une seule instruction par ligne) s'appelle une **séquence d'instructions**.

➢ Par analogie langagière :

Les valeurs, variables et expressions sont l'équivalent des sujets.

Les opérateurs, fonctions et méthodes sont l'équivalent des *verbes.*

L'ensemble des deux forment des instructions unitaires qui sont l'équivalent de *phrases.*

Une suite d'instructions est l'équivalent d'un *paragraphe.*

➢ Exemple :

Le programme (inintéressant) ci-contre est formé d'une seule séquence elle-même constituée de *2* instructions.

```
1  print(1+2)
2  2//3
```

La 1ère instruction est composée de la fonction *print( )* qui agit sur l'expression *1 + 2*.

La 2ème instruction est composée de *l'opérateur // qui agit entre les 2 opérandes 2 et 3.*

B. Présentation et syntaxe des instructions et séquences d'instructions :

Contrairement à d'autres langages qui délimitent les instructions et blocs d'instructions par des signes, Python le fait par la mise en page. D'où la clarté d'un programme écrit en Python !

Ce qui oblige à être très rigoureux dans la présentation. ⇒ Très bonne habitude d'apprentissage ! 😉

1. Instructions séparées par retours à la ligne :

En langage C par exemple, les instructions sont séparées syntaxiquement par des « ; ».

En langage Python, les instructions sont séparées par seulement des retours à la ligne !

D'où obligatoirement une seule instruction par ligne (ce qui n'est pas obligatoire en C, mais conseillé !).

2. Indentation : définition.

- **Indenter une ligne de code signifie décaler (tabuler) cette instruction vers la droite.**
- En Python, par convention, l'indentation conseillée est de 4 espaces.

3. Instructions composées :

➢ Définition :

Les instructions composées sont les boucles, tests conditionnels et fonctions (voir cours 2 Python).

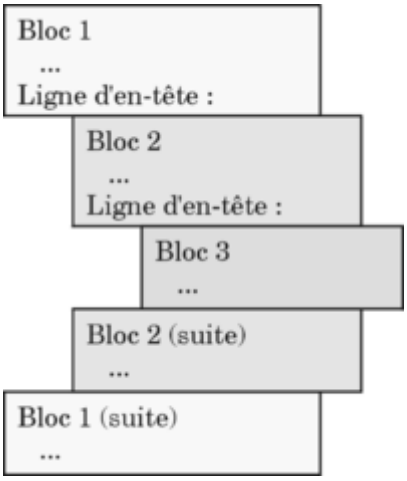
➢ Structure d'une instruction composée :

Toutes les instructions composées ont toujours la même structure :

- une ligne d'en-tête terminée par « : ».
- sous cette ligne d'en-tête : une ou plusieurs instructions indentées (décalées) au même niveau.

```
Ligne d'en tête :
    première instruction
    ... ...
    ... ...
    dernière instruction
```

4. Blocs d'instructions :



➢ Définition :

Un bloc d'instructions est une séquence d'instructions alignées sur la même indentation.

Les blocs d'instructions sont créés par les instructions composées.

➢ Organisation des blocs :

Comme les instructions composées peuvent encore se composer entre elles, il y a souvent plusieurs blocs indentés à plusieurs niveaux.

Toutes les instructions d'un bloc doivent être indentées exactement au même niveau : pas un espace en plus ou en moins sinon IdentationError !

5. Quels espaces et tabulations sont ignorés à l'exécution ?

A part ceux qui servent à l'indentation ou ceux à l'intérieur d'une chaîne de caractères, les espaces et tabulations placés à l'intérieur des instructions sont ignorés à l'exécution (tout comme les commentaires).

6. Exercice : instructions composées ; indentations.

Indiquer toutes les erreurs d'indentation et de syntaxe dans les schémas suivants (les points sont des espaces) :

En rouge les erreurs de « : » manquants et les erreurs d'indentation.

```
Instruction1
En-tête 1 :
Instruction bloc1
Instruction bloc1
En-tête 2 :
....Instruction bloc2
..........Instruction bloc2
....Instruction2
```

```
....Instruction1
En-tête 1 :
....Instruction bloc1
..........Instruction bloc1
........En-tête 2 :
..............Instruction bloc2
........Instruction bloc2
........Instruction2
```

Ai-je tout compris ? Instructions.	☹	😐	☺	☺☺
Définitions Instructions, Séquence d'instructions, Bloc d'instructions.				
Définition Indentation.				
Comprendre la structure en blocs de lignes de code.				
Corriger les erreurs d'indentation et de syntaxe des instructions composées.				

VIII. INSTRUCTIONS DE BASE.

Voyons maintenant quelques instructions simples mais incontournables en Algorithmique-Programmation.

A. Affectation d'une valeur à une variable :

La toute première instruction utile est de remplir avec une valeur la petite boîte représentant la variable.

1. Définition de l'affectation :

• **L'affectation d'une valeur à une variable est l'instruction basique qui permet d'attribuer une valeur à une variable : variable ← valeur**

Par abus de langage, on dit souvent « affectation de variable » ou « assignation de variable ».

• En langage Python, cette assignation s'effectue avec l'opérateur d'affectation « = », selon la syntaxe :

(à gauche) nom_de_variable = expression à évaluer (à droite)

L'expression à évaluer (à calculer) à droite du signe = peut être juste une simple valeur comme une expression très complexe combinant calculs, valeurs retournées par d'autres variables ou fonctions etc.

2. Différents types d'affectations :

Types d'affectation	*Exemples*	*Commentaires*
Affectation d'une quantité	nb_bisous = 3 + 2	la valeur 3 + 2 (c-à-d 5) est injectée dans la case mémoire nommée nb_bisous.
Affectation d'un texte	• matiere = "Info" • couleur = 'bleu' + 'vert'	Chaîne de caractères écrite soit entre "guillemets", soit entre 'apostrophes' droits.
Affectations parallèles	a , n = 2.5 , "bleu"	Equivaut à n = "bleu" puis a = 2.5
Auto-affectation	• k = k ** 2 • *t* = 3 * t – 2	**Une expression avec la valeur de la variable est calculée puis réaffectée à la variable elle-même.**
Incrémentation **Décrémentation**	• k = k + 2 • compteur = compteur – 2	**Auto-affectation additionnée ou soustraite d'un certain pas (d'un certain nb) entier.**

Exemples d'itérations (incrémentations) - décrémentations		
variable acc incrémentée d'un pas 3 acc = *acc + 3*	variable t décrémentée d'un pas k *t = t – k*	variable total décrémentée de n *total = total – n*

3. Ne pas confondre affectation et égalité mathématique !

➢ **Attention, l'opérateur « = » n'a donc pas le même sens en Python qu'en Mathématiques !**

Cela provoque de nombreuses incompréhensions de la part des apprenants débutants.

Exemple :

```
1  x = 25
2  x = x + 2
```

ne se traduit pas par

```
1  x = 25
2  25 = 25 + 2
```

En effet, à la ligne 2, il y a d'abord évaluation par la droite de l'expression x + 2. Le résultat 27 de cette évaluation est alors « versé » dans la variable x. Quelle est la valeur contenue alors par x ? *27.*

➢ Il y a donc une double difficulté syntaxique dans l'affectation de valeur à une variable :

1. Le signe « = » représente en fait une flèche d'attribution de la droite vers la gauche : ←.

   D'ailleurs, en algorithmique, le schéma d'affectation s'écrit : variable ← expression à évaluer.

2. La lettre « x » peut représenter 2 choses différentes dans une même instruction d'affectation !

   A droite du signe =, x représente la *valeur* de la variable x en tant que contenu.

   A gauche du signe =, x représente la variable x en tant que contenant.

4. Exercices sur l'affectation :

❶ Pour les 4 scripts (petits programmes) suivants, que s'affichera-t-il à l'écran ?

Note : L'instruction print ('a = ', a, 'et b = ', b) affiche sur une même ligne de l'écran le texte « a = » suivi de la valeur de la variable a puis le texte « et b = » suivi de la valeur de la variable b puis retour à la ligne.

```
1  a, b = 'Mahe', 'Moeve'
2  print ('a = ',b,'et b = ',a)
```

a = Moeve et b = Mahe

```
1  a, b = Mahe, Moeve
2  print ('a = ',a,'et b = ',b)
```

NameError : Mahe et Moeve sans '' sont des variables non définies

```
1
2  a = b
3  b = 1
4  print ('a = ',a,' et b = ',b)
```

NameError : la variable b n'est pas définie !

```
1  a, b = 2, 3
2  a = b
3  b = a
4  print ('a = ',a,' et b = ',b)
```

a = 3 et b = 3 : pas d'échange des valeurs !

❷ Sans affectations parallèles, écrire un script permutant les valeurs de 2 variables a et b.

`a , b = 2 , 3` Vérifier votre script avec le tableau d'état des variables ci-contre.

Solution avec variable auxiliaire

1. *aux = a*
2. *a = b*
3. *b = aux*

Solution sans variable auxiliaire

1. *a = a + b*
2. *b = a – b*
3. *a = a – b*

	état des variables		
	a	b	aux
	Pour le 1er script :		
1	a	b	a
2	b	b	a
3	b	a	a
	Pour le 2ème script :		
1	a + b	b	
2	a + b	a	
3	b	a	

5. Initialisation d'une variable :

➢ Lorsqu'une variable est déclarée, une zone mémoire est réservée et nommée.

Mais que contient alors cette zone ? Ce peut être rien comme ce peut être aussi hélas *n'importe quoi* !

Comme la variable peut contenir au départ rien ou n'importe quoi, il faut lui assigner une valeur de départ : on appelle cette 1ère assignation « initialiser la variable ».

Définition : L'initialisation d'une variable est un cas particulier de l'affectation d'une variable.

L'initialisation d'une variable est la toute 1ère affectation de cette variable.

Cette initialisation est parfois écrite en début de programme.

➢ Comme déjà dit plus haut, il n'y a pas en Python d'instructions spécifiques pour la déclaration et/ou le typage des variables comme en Pascal ou en C (voir France IOI correction d'un exo en Pascal ou en C).

En Python, la déclaration et le typage se font à la volée au moment de l'initialisation de manière dynamique. Cela constitue une force et en même temps une faiblesse de Python : concision dans l'écriture des programmes mais moindre rigueur dans la structuration.

Exemple : Que va-t-il s'afficher lors de l'exécution du script suivant ? Pourquoi ?

```
1  a = 3
2  b = a + c
```

NameError : la variable c n'est pas définie. Elle n'a pas été initialisée !

Attention, dans certains langages, même si on oublie d'initialiser cette variable c, le programme pourrait très bien fonctionner (la variable c serait alors initialisée par son contenu par défaut, c-à-d n'importe quoi !) mais fournirait alors des résultats complètement inattendus.

➢ Un programme ne peut se contenter de ne manipuler que des variables ! Il faut qu'il puisse être en interaction avec ses utilisateurs, l'environnement ou d'autres programmes etc. Soit en recevant des données, on parle alors d'entrée (de données). Soit en renvoyant des données, on parle alors de sortie (de données).

L'une des premières nécessitées quand on programme est de pouvoir afficher des trucs à l'écran.

B. Instruction de sortie print( ) :

1. Hello world !

La tradition veut que le tout premier programme de tout apprenti programmeur affiche « Hello world ! ».

L'instruction à écrire est : `print ('Hello world !')`

Cette instruction est constituée de la fonction *print( )* et de la chaîne de caractères *'Hello world !'*.

2. La fonction prédéfinie print( ) :

➢ Nous avons déjà vu la fonction print( ) lors de l'étude des chaînes de caractères p.11.

- **L'instruction de sortie print( ) :**

  prend en entrée (entre parenthèses) un ou plusieurs textes ; variables, expressions à évaluer.

  affiche en sortie (à l'écran en général) une chaîne de caractères concaténée + un retour à la ligne.

- Syntaxe complète en entrée : **print(objet1 , objet2 , etc. , *sep=' ' , end = ' ' , file = ' '*)**

objet1, objet2, etc. peuvent être soit un texte soit une variable (n'importe quel type) soit une expression à évaluer. Séparation des objets comme d'habitude par des virgules.

sep = ' ', end = ' ' , file=' ' paramètres facultatifs pour changer le comportement par défaut du print( ).

- Comportement détaillé du print( ) par défaut (sans aucun paramètre spécifié) :

print (objet1 , objet2 , objet3 , etc.) → "valeur1""valeur2""valeur3"etc.

Evaluation de tous les objets + Conversion en chaîne de caractères des valeurs des objets.

Concaténation (accolement) des chaînes de caractères précédentes. **Séparation par défaut : 1 espace**.

Affichage de la chaîne de caractères concaténée. **Fin de ligne par défaut = 1 retour à la ligne.**

Remarques : La fonction print( ) ne retourne aucune valeur mais affiche juste du texte à l'écran.

Cette syntaxe permet d'écrire plusieurs print( ) en un seul.

➢ Exemple d'un print( ) sans paramètres :

```
a, b, c = 'Je t\'aime', 2 , 1>0
print(a, b,'fois ?',c,'!')
```

→ *Je t'aime 2 fois ? True !*

➢ Application du print( ) par défaut : Qu'affichent les scripts suivants ? Vérifiez sur la console si besoin.

prenom = 'Loulou' conj = 'et' print(prenom, conj, 'Boutin') *Loulou et Boutin*	a , b = 1 , a > B print("Vrai vaut", a, "c'est", b) *Vrai vaut 1 c'est True*	a , b = 2 , 3 print("la somme vaut", a + b, '.') *la somme vaut 5.*
a , k ="I love" , "Maths." print(2*(a + k)) *I loveMaths.I loveMaths.*	a, b , c = 'ca', 2 , 'pi' print ('Vulgaire :' , b*a +b*c) *Vulgaire : cacapipi*	a , b , c , d = 2 , 3 , B > a , B< a a , b = b , a print ('valeurs échangées ?', *d*) *valeurs échangées ? True*

3. Utilisation des paramètres optionnels de la fonction print( ) :

Par défaut, le séparateur est *1 espace blanc* et la terminaison *1 retour à la ligne.*

Comment changer ces comportements par défaut des paramètres de séparation, de terminaison ou de sortie ?

Paramètre	*Nom*	*Exemples à vérifier à la console : instruction → résultat*	*Commentaires*
sep	Séparateur	•print('ab','cd','ef',sep='k,') → abk,cdk,ef • print('ab','cd','ef',sep='') → abcdef	• Les caractères séparateurs remplacent le 1-espace par défaut. • Si sep= vide, il y a accolement.
end	Terminaison	• print('ab','cd','ef',end='k,') → abcdefk, • print('ab','cd','ef',end='') → abcdef	• La terminaison remplace le retour à la ligne par défaut. • Si end= vide, pas de retour ligne.
file	Sortie	fh = open("data.txt","w") print('ab','cd','ef',file=fh) fh.close( )	file= change la sortie standard qui est l'écran en général. Ici, impression dans le fichier fh.

➢ Application : Ecrire 1 seule instruction qui affichera :

Affichage à l'écran	*Terminaison*	*Instruction (vérifier à la console ou sur pythontutor)*
a . b . c . d	pas de retour ligne	*print('a' , 'b' , 'c' , 'd' , sep = '.' , end = '')*
1'2'3	une tabulation	*print(1 , 2 , 3 , sep = '' , end = '\t')*
1 2 3	pas de retour ligne après le 3.	*print(1 , 2 , 3 , sep = '\n' , end = '')*

4. Trois différentes façons de formater la sortie de la fonction print( ) :

Voici 3 scripts affichant exactement la même chose mais avec un print( ) selon 3 formats différents.

classique	*avec la méthode .format( )*	***avec f"strings" (Python ≥ 3.6)***
nom, age = "Célia", 27 print (nom , "a" , age , "ans.")	nom, age = "Célia", 27 print ("{ } a { } ans.".format (nom,age))	**nom, age = "Célia", 27** **print (f"{nom} a {age} ans.")**

1. Quelle phrase commune est affichée par ces 3 scripts ? *Célia a 27 ans.*
2. Quel formatage est le plus naturel et le plus pratique ? *Avec f"strings" !*

Les f"strings" sont la façon moderne d'écrire un print(plusieurs arguments). Ils allient les avantages de la méthode classique (intuitif) avec ceux de la méthode format (possibilités étendues de formatage).

Dorénavant, on utilisera les f"strings" dès qu'on aura besoin d'un print( ) un peu compliqué.

Il existe évidemment d'autres instructions de sortie : par exemple pour enregistrer des données dans un fichier (voir cours 7) ou dans une base de données, pour renvoyer des données vers des capteurs, etc.

C. Instruction d'entrée input( ) :

➢ Prenons l'exemple d'une variable nb_bisous qu'on veut initialiser dès le début d'un programme :

- On peut l'initialiser à l'intérieur même du programme par l'instruction nb_bisous = 5.

L'inconvénient est que la valeur de départ de nb_bisous sera figée. Et seuls ceux qui ont accès au corps du programme (concepteurs, programmeurs etc.) peuvent la changer en modifiant le programme.

- On peut l'initialiser à l'extérieur du programme, sans modifier le programme.

Cette fois-ci, n'importe quel utilisateur du programme pourra entrer le nombre de bisous qui lui correspond !

- **La fonction input( ) permet à l'utilisateur d'entrer au clavier une donnée.**
  Par défaut, la fonction input( ) renvoie des données de type caractère (str).
- Pour que la donnée entrée au clavier soit considérée autrement que comme du texte, on utilise la syntaxe :
  variable = type (input("phrase introductrice ultra conseillée")) où type est le type désiré.
- La phrase introductrice est dans les faits obligatoire sinon, on ne voit pas la demande d'entrée à l'écran !

➢ Exemples :

x = int (input()) donnée entrée au clavier convertie en type *entier*, rangée dans la variable *x.*

p = float (input()) donnée entrée au clavier convertie en type *float* (pseudo réel), rangée dans *p.*

aux = input() donnée entrée au clavier de type par défaut *character (str),* rangée dans *aux.*

nb_pieces_or = int(input("Entrer le nombre de pièces d'or :")) donnée convertie en type *entier*, rangée dans *nb_pieces_or*, précédée de la phrase introductrice . *"Entrer le nombre de pièces d'or :".*

➢ Application : Ecrire les instructions d'entrée au clavier avec les paramètres suivants.

Phrase introductrice	*Type*	*Variable*	*Instruction (vérifier à la console ou sur pythontutor)*
Amie préférée :	caractère	personne	personne = input(' Amie préférée : ')
Nb d'amours ?	entier	nb_amours	nb_amours = int(input('Nb d\'amours ?'))
Tu m'adores ?	booléen	réponse	réponse = bool(input("Tu m'adores ?"))

➢ Exercice sur entrée-sortie :

Ecrire un script qui en entrée demande si tu m'aimes et qui réagit en sortie en fonction de la réponse tapée.

```
réponse = bool(input("Tu m'aimes ?"))
if réponse :    # pas besoin d'écrire réponse == True.
    print('Super !')
else :
    print('Sniff…')
```

Il existe évidemment d'autres instructions d'entrées : par exemple pour lire des données dans un fichier (voir cours 7) ou dans une base de données, ou pour collecter des données depuis des capteurs, etc.

Ai-je tout compris ? Instructions de base.	☹	😐	☺	☺☺
Qu'est-ce que l'affectation de variable ?				
Différents types d'affectations : affectations parallèles ; auto-affectation ; incrémentation etc.				
Pourquoi faut-il initialiser les variables ?				
Script qui échange les valeurs de 2 variables.				
Fonction prédéfinie print( ) : comportement détaillé par défaut.				
Utiliser les paramètres optionnels de la fonction print( )				
Fonction prédéfinie input( ) : comportement par défaut.				
Ecrire une instruction d'entrée avec un texte introducteur et en imposant le type.				

Quel est le sujet du prochain cours Python ? *Structures de contrôle (I) : Boucles et Tests.*