1NSI	PROGRAMMATION	S2
	Introduction à la programmation en Python	

En **python** tout est **objet** :

- En informatique, un **objet** est une zone en mémoire (RAM) contenant toutes les informations (caractéristiques et comportement) d'une donnée.
- Un **objet** (on dit aussi **instance**) est créé dans l'espace d'exécution à partir d'un **modèle**, appelé **classe**, qui définit le type de la donnée et qui hérite de toutes les informations concernant la donnée.

Les classes Python sont nombreuses. En voici quelques exemples dans ce tableau :

Classe	Signification
`int`	Entier relatif
`float`	Réel
`complex`	Complexe
`bool`	Booléen
`str`	Chaîne de caractères
`list`	Liste
`tuple`	Tuple
`dict`	Dictionnaire
`set`	Ensemble
`function`	Fonction
`type`	Classe (définie par l'utilisateur)
...	...

I. VARIABLES PYTHON

Nous avons vu précédemment qu'un programme traite des données rangées dans des cellules de la mémoire (RAM). Chaque cellule de la mémoire a une adresse, ce qui permet au processeur de savoir où sont rangées les données pour éventuellement modifier leur valeur au cours de l'exécution d'un programme.

Une **variable** est donc simplement une donnée qui peut *changer de valeur*.

- Elle est identifiée grâce à un **nom** et est associée à une **adresse mémoire**.
  - Python étant un langage de haut niveau, l'adressage d'une variable en mémoire est complètement transparent pour le programmeur. C'est l'interpréteur Python qui s'en charge lors de l'exécution du programme.

I.A Règles de nommage d'une variable

Le nom d'une variable Python doit obéir à quelques règles simples pour des raisons de lisibilité.

1. Un nom de variable est une **séquence de lettres** (a → z, A → Z) et **de chiffres** (0 → 9) qui doit toujours commencer par une lettre.
2. Un nom de variable doit être **explicite**.
   - Hors contexte, les noms `x, y, a, toto...` sont donc à proscrire.
3. Un nom de variable ne doit pas contenir de lettres accentuées, cédilles, espaces et caractères spéciaux ($, #, @, -, etc.), à l'exception du caractère _ dit caractère souligné, ou underscore, ou tiret du 8.
4. Dans le cas d'un nom de variable devant contenir plusieurs mots, Python préconise que chaque mot soit écrit en minuscule et séparé par le caractère souligné (underscore, tiret du 8)

Exemples :

```
pression, temperature, vitesse…
pression_atmospherique, temperature_degres_celsius,
vitesse_de_propagation…
```

Remarques :

De nombreux langages préconisent la technique **camelCase**. Elle peut être utilisée en Python, mais ce n'est pas l'usage :

- Le terme camelCase fait référence aux bosses et creux d'un dos de chameau représentant l'alternance de majuscules et de minuscules entre les premières lettres des mots utilisés et les suivantes.

Exemples :

```
pressionAtmospherique, temperatureDegresCelsius, vitesseDePropagation
```

La casse est significative, les caractères majuscules et minuscules sont distingués.

Exemples :

`temperature` et `Temperature` sont deux variables différentes.

I.B Affectation d'une variable

Python est un langage **dynamiquement typé**. Il n'est pas nécessaire de déclarer le type des variables avant de les utiliser contrairement aux langages statiquement typés tel que C, C++, Java...

Le typage d'une variable s'opère au moment de l'**affectation** (ou de l'**assignation**) d'une valeur à une variable grâce à l'opérateur '=', en deux temps :

1. **Évaluation** (calcul) de la partie droite de l'opérateur d'affectation (selon des règles de priorité).
2. **Rangement** du résultat dans la partie gauche de l'opérateur d'affectation.

Exemple :

L'instruction Python suivante est une affectation simple :

```
a = 12
```

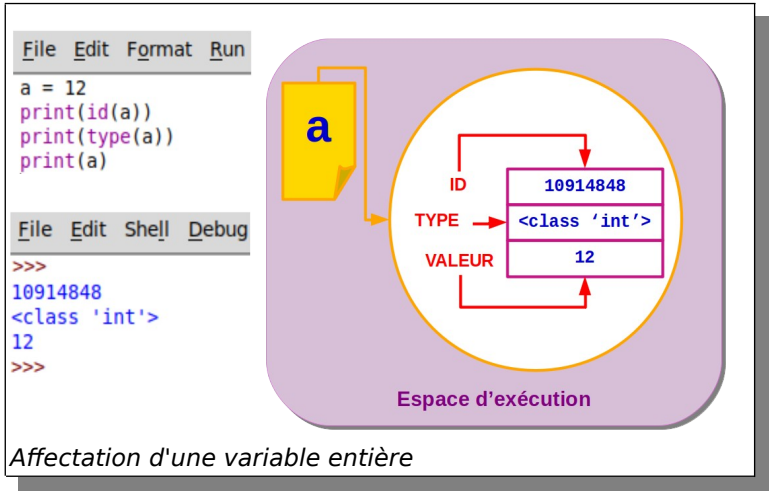
Elle affecte à une variable 'a' la valeur entière 12.

De façon schématique, l'interpréteur Python complète dans l'espace d'exécution, c'est à dire la mémoire (RAM) :

- Une étiquette 'a' pour la variable, en relation avec une adresse mémoire de la (RAM) qu'il est le seul à connaître (transparent pour le programmeur)
  - et dans laquelle il a stocké la donnée (valeur entière) 12.
- Des zones mémoires dans lesquelles il stocke des informations supplémentaires concernant la donnée.

Par exemple, les instructions Python suivantes :

- `id(a)`
  - permet de récupérer l'identité (ID) de la variable 'a', image de l'adresse mémoire où est stockée la donnée, mais ce n'est pas l'adresse à proprement parler.
- `type(a)`
  - permet de récupérer le type de la variable 'a'.
- `help(a)`
  - permet de connaître les caractéristiques et les comportements attendus de la variables 'a'.



Affectation d'une variable entière

***Attention** : Ne pas confondre l'opérateur d'affectation '=' avec le symbole d'égalité mathématique, qui est en python '==' (2 fois le signe égal).*

I.C Mots réservés

Il est **interdit** d'utiliser comme nom de variable les **mots** ci-dessous car ils sont réservés au langage :

and	as	assert	break	class	continue	def
del	elif	else	except	False	finally	for
from	global	if	import	in	is	lambda
None	nonlocal	not	or	pass	raise	return
True	try	while	with	yield		

I.D Exercices

1 Dans un interpréteur Python **saisir** les instructions d'affectation de variables suivantes et les **exécuter** puis **compléter** le tableau.

Instruction	ID	TYPE	VALEUR
`b = 7`			
`c = -8`			
`d = 5.0`			
`e = -6.8`			
`f = 7e-3`			
`g = -10e3`			
`h = 7+7j`			
`i = 8+0j`			
`j = True`			
`k = False`			
`m = "Hello"`			
`n = 'Hello'`			
`p = """Hello"""`			

2 **Compléter** la phrase suivante :

Les ID ont toutes une valeur ... car ces variables sont toutes des objets ... .

A retenir :

- Les types numériques sont dit **signés** car leurs valeurs peuvent être positives ou négatives,
- Un réel peut être écrit sous forme exponentielle,
- Il n'existe pas de type 'caractère' en Python.

I.E Affectations multiples

Python permet également d'effectuer des *affectations simultanées*.

Dans votre interpréteur Python **saisir** les instructions suivantes et les **exécuter** puis **compléter** le tableau :

Instruction	Résultat
`x = y = 5.2`	
`x`	
`type(x)`	
`y`	
`type(y)`	

I.F Affectations parallèles

Python permet d'effectuer des *affectations parallèles* à l'aide d'un seul opérateur :

Dans votre interpréteur Python **saisir** les instructions suivantes et les **exécuter** puis **compléter** le tableau :

Instruction	Résultat
`a, b , c = 4, 8.33, "Hello"`	
`a`	
`b`	
`c`	
`type(a)`	
`type(b)`	
`type(c)`	

I.G Exercices

1 **Écrivez** l'instruction qui affecte les valeurs respectives 3, 4, 5 à trois variables a, b, c avec un seul opérateur '='.

2 **Écrivez** l'instruction qui initialise trois variables x, y, z simultanément à zéro.

3 **Affectez** dans un interpréteur Python les valeurs respectives 7, 33, 45 à trois variables h, m, s.

II. ÉCRITURE DES NOMBRES

Pour faciliter l'écriture et la lecture des nombres aux programmeurs, les nombres sont le plus souvent écrits dans la **base 10**, ou base **décimale**.

Mais ce n'est pas la seule façon de représenter des nombres en informatique. En fonction des besoins, la base **hexadécimale** (**base 16**), la base **binaire** (**base 2**) et même la base **octale** (**base 8**) sont couramment utilisées pour coder des nombres.

Par exemple l'entier 10 en notation **décimale** s'écrit :

- `0xA` ou `0xa` en notation **hexadécimale**,
- `0o12` en notation **octale**,
- `0b1010` en notation **binaire**.

II.A Conversion des nombres

Pour convertir une valeur numérique **entière** en **binaire**, **hexadécimal**, **octal**, on utilisera les instructions Python (fonctions built-in) suivantes :

- **bin**()
- **hex**()
- **oct**()

Attention :

- Le résultat renvoyé par ces instructions est une chaîne de caractères.
- **Taper** **help**(*nom de l'instruction)* pour obtenir l'aide de Python.

II.B Exercices

1 **Dans** un interpréteur Python **saisir** les instructions d'affectation de variables suivantes et les **exécuter** puis **compléter** le tableau.

Instruction	TYPE	VALEUR
`a = 0xFF`		
`b = 0b1111`		
`c = 0o777`		

2 Dans un interpréteur Python **saisir** les instructions de conversion suivantes et les **exécuter** puis **compléter** le tableau.

Instruction	Résultat obtenu	Type du résultat
`bin(10)`		
`oct(10)`		
`hex(10)`		
`bin(0xA)`		
`bin(0xF)`		
`bin(0xFF)`		
`hex(255)`		
`hex(65535)`		
`hex(0o10)`		
`oct(0xFF)`		
`oct(8)`		

Voir cours transcodage

III. LES OPÉRATEURS

Les opérateurs permettent de réaliser des calculs sur des variables.

III.A Les opérateurs mathématiques

Opérateur	Nom	Effet
`=`	*Affectation*	*Affecte une valeur à une variable*
`+`	Addition	Ajoute deux valeurs
`-`	Soustraction	Soustrait deux valeurs
`*`	Multiplication	Multiplie deux valeurs
`/`	Division décimale	Divise deux valeurs (Résultat réel)
`//`	Division entière	Calcule le quotient de la division euclidienne
`%`	Modulo	Calcule le reste de la division euclidienne
`**`	Puissance	Élève à la puissance

III.B Les opérateurs de comparaison (de test)

Opérateur	Dénomination	Rôle	Résultat retourné
`==`	opérateur d'égalité	Compare deux valeurs et vérifie leur égalité	True (vrai) si oui, False (Faux) si non.
`!=`	opérateur de différence	Vérifie qu'une variable est différente d'une valeur	
`<`	opérateur d'infériorité stricte	Vérifie qu'une variable est strictement inférieure à une valeur	
`<=`	opérateur d'infériorité	Vérifie qu'une variable est inférieure ou égale à une valeur	
`>`	opérateur de supériorité stricte	Vérifie qu'une variable est strictement supérieure à une valeur	
`>=`	opérateur de supériorité	Vérifie qu'une variable est supérieure ou égale à une valeur	
`is`	Comparaison de 2 objets	Compare si 2 objets sont en réalité un et un seul et même objet	
`is not`	Comparaison de 2 objets	Compare si 2 objets sont deux objets différents	

III.C Les opérateurs d'assignation

Opérateur	Exemple	Instruction équivalente
+=	a += 1	a = a + 1
-=	a -= 2	a = a - 2
*=	a *= 3	a = a * 3
/=	a /= 3	a = a / 3

III.D Les opérateurs logiques

Opérateur	Fonction logique équivalente
or	OU logique
and	ET logique
not	NON logique, complémentation ou inversion logique

Voir cours logique booléenne

III.E Les opérateurs logiques binaires (bitwise : bit à bit)

Opérateur	Dénomination	Exemple	Résultat
|	OU logique bitwise	0 | 1	1
&	ET logique bitwise	0 & 1	0
^	OU Exclusif logique bitwise	1 ^ 1	0
<<	Décalage à gauche (multiplication par 2)	1 << 2	4
>>	Décalage à droite (division par 2)	4 >> 2	1

Ces opérateurs traitent les nombres entiers sur lesquels ils agissent comme des séquences de bits (nombres binaires). Les opérateurs logiques sont alors appliqués sur les bits de même rang :

- `3 & 2 = 0b11 & 0b10 = 0b10 = 2`

A retenir :

Lors de l'utilisation des divers opérateurs, il est préférable de placer des parenthèses pour s'assurer du résultat souhaité car il existe des priorités sur les opérateurs.

III.F Exercices

1 **Affectez** la valeur entière 7 à la variable x, puis **exécutez** dans un interpréteur Python les instructions suivantes :

Instruction	Résultat affiché après exécution de l'instruction
x + 3	
x - 3	
x * 3	
x / 3	
x // 3	
x % 2	
x ** 2	

2 Quel est le type du résultat d'une addition d'un int et d'un float ?"

Instruction	Résultat
type(2+1.2)	

3 Que produisent les instructions suivantes, si a=2, b=3, c=4 ? **Prévoyez** le résultat avant d'exécuter l'instruction dans un interpréteur Python.

Instruction	Nouvelle valeur de a
a-=b	
a+=(b+c)	

4 **Affectez** la valeur entière 7 aux variables x et y, puis **exécutez** dans *un interpréteur Python* les instructions suivantes :

Instruction	Résultat
y == 7	
y == 3	
y != 3	
y < 3	
y <= 7	
y > 7	
y >= 7	

5 **Utilisez** un interpréteur Python pour compléter les tables de vérité des fonctions logiques de bases OU, ET, NON.

Instruction	Fonction logique OU
`0 or 0`	
`0 or 1`	
`1 or 0`	
`1 or 1`	

Instruction	Fonction logique ET
`0 and 0`	
`0 and 1`	
`1 and 0`	
`1 and 1`	

Instruction	Fonction logique NON
`not 0`	
`not 1`	

Voir cours logique booléenne

6 Que produisent les instructions suivantes ?

Instruction	Résultat
`True and True`	
`False and True`	
`False or True`	
`3 and 2`	
`3 or 2`	
`3 or 0`	
`3 and 0`	
`not 0`	

7 Que produisent les instructions suivantes ? **Prévoyez** le résultat avant exécution.

Instruction	Résultat
`1 & 1`	
`1 & 0`	
`9 & 8`	
`0b1001 & 0b1000`	
`1 | 1`	
`1 | 0`	
`9 | 8`	
`0b1001 | 0b1000`	
`0xa & 0xf`	
`0xa | 0xf`	
`0b1001 << 1`	
`0b1001 >> 1`	
`0b1001 | 0b1000`	

8 Soit les variables a = 5 et b = 2. Que produisent les instructions suivantes ? **Prévoyez** le résultat avant exécution.

Instruction	Résultat obtenu
`a==5 and b==2`	
`a==5 & b==2`	
`(a==5) & (b==2)`	
`a==5 and b<2`	

IV. LES FONCTIONS

La notion de fonctions en informatique est comparable à la notion de fonctions mathématiques :

Exemple d'une fonction mathématique :

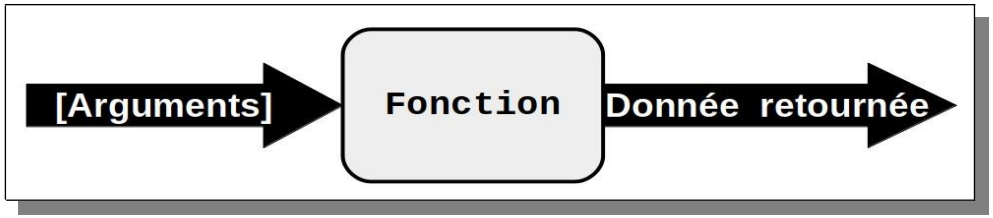
- y = f(x) = 2x+1
  - Si x = 4 alors y = 2 x 4 + 1 = 9

D'un point de vue informatique, une approche efficace d'un problème complexe consiste, pour le programmeur, à le décomposer en plusieurs sous-problèmes plus simples, sous la forme de fonctions.

La décomposition en fonctions d'un programme présente au moins deux avantages :

- Elles sont étudiées séparément, ce qui facilite la **mise en œuvre** et la **maintenance** d'un programme.
- Elles peuvent être éventuellement utilisées à plusieurs reprises dans un programme, on parle alors de **factorisation** du code.

Une fonction reçoit en **entrée** des arguments (ou paramètres) qui ne sont pas toujours obligatoires, exécute un travail avec ces arguments (quand ils existent) et retourne une donnée en **sortie** :



IV.A Définir une fonction en Python

```
def nom_de_la_fonction(arguments) :
        """docstring"""
        <bloc d'instructions>
        return expression
```

Exemple :

```
def f(x) :
        """y = 2x+1"""
        y = 2 * x + 1
        return y
```

IV.B Appeler une fonction en Python

On appelle une fonction en invoquant son nom dans une instruction, la valeur de retour de la fonction est mémorisée dans une variable.

Résultat obtenu dans un interpréteur

```
>>> def f(x) :
        """y = 2x+1"""
        y = 2 * x + 1
        return y

>>> solution = f(2)
>>> solution
5
```

IV.C Exercices

1 Quel est le résultat renvoyé par la fonction ci-dessous si l'on exécute dans l'interpréteur : `solution = g(2, 5, 3)` ?

```
def g(a, x, b) :
        """y = ax+b"""
        y = a * x + b
        return y
```

2 Quel est le type de la variable solution ?

3 **Codez** en Python la fonction $y = x^2 + 2x + 1$

4 Quel est le résultat renvoyé par la fonction ci-dessous si l'on exécute dans l'interpréteur : `dire_bonjour("Cover", "Hary")` ?

```
def dire_bonjour(nom, prenom):
        phrase = f"Bonjour {prenom} {nom}."
        return phrase
```

5 Quel est le type de la valeur de retour ?

6 Quel est le résultat renvoyé par la fonction ci-dessous si l'on exécute dans l'interpréteur : `dire_bonjour("Cover", "Hary")` ?

```
def dire_bonjour(nom, prenom):
        phrase = f"Bonjour {prenom} {nom}."
        print (phrase)
```

IV.D Fonctions prédéfinies

Appelées également fonctions *built-in*, elles sont intégrées dans Python :

```
print(), input(), range()...
```

IV.E Fonctions de module

Ces fonctions sont regroupées par thème, dans un fichier séparé que l'on appelle **module** (ou **librairie** ou **bibliothèque**).

Il existe de nombreux modules :

- intégrés à Python comme le module `math` par exemple,
- développés par des tierces personnes, ce qui nécessite parfois une installation dans le système avant utilisation.

Le module `math`, contient les définitions de nombreuses fonctions mathématiques telles que *sinus*, *cosinus*, *tangente*, *racine carrée* mais aussi la constante *pi*. Pour pouvoir utiliser le module, il suffit de l'importer dans le script.

1 Importer un module entier

Pour importer le module `math` par exemple, il suffit d'écrire l'instruction suivante :

```
import math
```

La totalité du module est mis en mémoire de l'ordinateur. Pour accéder au contenu du module, il faut faire précéder la fonction ou la constante par le nom du module suivi de l'opérateur point '.'

`math.pi`	calcule 3.141592653589793
`math.pow(x, a)`	calcule x à la puissance a
`math.cos(x)`	calcule le cosinus de l'angle x.
`math.sinus(x)`	calcule le cosinus de l'angle x.
`math.sqrt(x)`	calcule la racine carrée de x

Pour d'autres fonctions mathématiques, consulter la documentation Python : https://docs.python.org/3/library/math.html

2 Importer tout ou partie d'un module

Si vous souhaitez importer *la totalité* des fonctions contenues dans le module de telle sorte qu'elles soient *directement accessibles* par simple appel de leur nom, il faut importer le module de la façon suivante :

```
from math import*
```

Dans ce cas, pour accéder la la constante *pi,* il suffit de saisir son nom :

```
pi
```

Mais, si vous ne souhaitez utiliser seulement que *cos()* et *pi* par exemple :

```
from math import cos, pi
```

Cette dernière méthode présente l'avantage de ne pas surcharger la mémoire de l'ordinateur pour rien.

3 Exercice

Quelles sont les valeurs des variables après exécution des instructions suivantes ?

Instruction	Résultat
`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)`	