

# C0 Un peu de Python

## 1. Types de base

### Types de base

Type	Opérations	Commentaires
<code>int</code>	<code>+</code> , <code>-</code> , <code>*</code> , <code>//</code> , <code>%</code>	Entiers signés ou non signés. Taille dynamique limitée par la mémoire

# C0 Un peu de Python

## 1. Types de base

### Types de base

Type	Opérations	Commentaires
<code>int</code>	<code>+, -, *, //, %</code>	Entiers signés ou non signés. Taille dynamique limitée par la mémoire
<code>float</code>	<code>+, -, *, /</code>	Représentation des nombres en virgule flottante (norme <code>ieee754</code> : mantisse sur 53 bits, exposant sur 11 bits). Fonctions élémentaires dans <code>math.h</code>

### Types de base

Type	Opérations	Commentaires
<code>int</code>	<code>+</code> , <code>-</code> , <code>*</code> , <code>//</code> , <code>%</code>	Entiers signés ou non signés. Taille dynamique limitée par la mémoire
<code>float</code>	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code>	Représentation des nombres en virgule flottante (norme <code>ieee754</code> : mantisse sur 53 bits, exposant sur 11 bits). Fonctions élémentaires dans <code>math</code>
<code>bool</code>	<code>or</code> <code>and</code> , <code>not</code> , <code>all</code> , <code>any</code>	Evaluations paresseuses des expressions.

# C0 Un peu de Python

## 1. Types de base

### Types de base

Type	Opérations	Commentaires
<code>int</code>	<code>+</code> , <code>-</code> , <code>*</code> , <code>//</code> , <code>%</code>	Entiers signés ou non signés. Taille dynamique limitée par la mémoire
<code>float</code>	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code>	Représentation des nombres en virgule flottante (norme <code>ieee754</code> : mantisse sur 53 bits, exposant sur 11 bits). Fonctions élémentaires dans <code>math</code>
<code>bool</code>	<code>or</code> <code>and</code> , <code>not</code> , <code>all</code> , <code>any</code>	Evaluations paresseuses des expressions.
<code>str</code>	<code>+</code> , <code>*</code>	Noté entre quotes ( <code>'</code> ) ou guillemets ( <code>"</code> ). Longueur avec <code>len</code>

### Définir une fonction en Python

Pour définir une fonction en Python :

### Définir une fonction en Python

Pour définir une fonction en Python :

- qui ne renvoie pas de valeur :

```
1  def <nom_fonction>(<arguments>):  
2      <instruction>
```

- qui renvoie une valeur :

```
1  def <nom_fonction>(<arguments>):  
2      <instruction>  
3      return <resultat>
```

### Instructions conditionnelles

- Sans clause `else`

```
1  if <condition>:  
2      <instructions>
```

Exécute les <instructions> si la condition est vérifiée.

### Instructions conditionnelles

- Sans clause `else`

```
1  if <condition>:  
2      <instructions>
```

Exécute les <instructions> si la condition est vérifiée.

- Avec clause `else`

```
1  if <condition>:  
2      <instructions1>  
3  else:  
4      <instructions2>
```

Cela permet d'exécuter les <instructions1> si la condition est vérifiée, sinon on exécute les <instructions2>.



### Boucles while

### Boucles while

- La syntaxe d'une boucle **while** en Python est :

```
1 while <condition>:  
2     <instruction>
```

Cela permet d'exécuter les <instructions> tant que la <condition> est vérifiée.

### Boucles while

- La syntaxe d'une boucle **while** en Python est :

```
1 while <condition>:  
2     <instruction>
```

Cela permet d'exécuter les <instructions> tant que la <condition> est vérifiée.

- On ne sait pas a priori combien de fois cette boucle sera exécutée (et elle peut même être infinie), on dit que c'est une boucle **non bornée**.

### Boucles for

### Boucles for

- Les instructions :

```
1  for <variable> in range(<entier>):  
2      <instructions>
```

créent une variable parcourant les entiers de 0 à <entier> (exclu).

### Boucles for

- Les instructions :

```
1  for <variable> in range(<entier>):  
2      <instructions>
```

créent une variable parcourant les entiers de 0 à <entier> (exclu).

- Les <instructions> indentées qui suivent seront exécutées pour chaque valeur prise par la variable.

### Boucles for

- Les instructions :

```
1   for <variable> in range(<entier>):  
2       <instructions>
```

créent une variable parcourant les entiers de 0 à <entier> (exclu).

- Les <instructions> indentées qui suivent seront exécutées pour chaque valeur prise par la variable.
- La boucle for permet donc de répéter un nombre prédéfini de fois des instructions, on dit que c'est une boucle bornée.

### Exemples

- Ecrire un programme Python permettant de calculer le PGCD  $d$  de deux entiers naturels  $a$  et  $b$  en utilisant l'algorithme d'Euclide.



### Exemples

- Ecrire un programme Python permettant de calculer le PGCD  $d$  de deux entiers naturels  $a$  et  $b$  en utilisant l'algorithme d'Euclide.
- Ecrire un programme Python permettant de vérifier la conjecture de Collatz pour les entiers inférieurs à un entier  $N$  donné. Le programme affichera aussi la valeur maximale atteinte durant les itérations. Par exemple pour  $N = 1000$ , le programme affiche :  
Conjecture vérifiée, maximum atteint 250 504.

### Exemples

- Ecrire un programme Python permettant de calculer le PGCD  $d$  de deux entiers naturels  $a$  et  $b$  en utilisant l'algorithme d'Euclide.
- Ecrire un programme Python permettant de vérifier la conjecture de Collatz pour les entiers inférieurs à un entier  $N$  donné. Le programme affichera aussi la valeur maximale atteinte durant les itérations. Par exemple pour  $N = 1000$ , le programme affiche :  
Conjecture vérifiée, maximum atteint 250 504.
- Ecrire un programme Python permettant de simuler une marche aléatoire dans le plan (déplacement aléatoire équiprobable dans les 4 directions cardinales). On pourra visualiser les déplacements grâce à la librairie `turtle` et se déplacer jusqu'à sortir d'un cercle de rayon donné.

### Correction exemple 1

- Version itérative

```
1 def pgcd(a,b):  
2     while b!=0:  
3         a,b = b, a%b  
4     return a
```

### Correction exemple 1

- Version itérative

```
1 def pgcd(a,b):  
2     while b!=0:  
3         a,b = b, a%b  
4     return a
```

- Version récursive

```
1 def pgcd(a,b):  
2     if b == 0:  
3         return a  
4     return pgcd(b,a%b)
```

## Les listes de Python

- Les listes de Python sont des structures contenant zéro, une ou plusieurs valeurs (pas forcément du même type).

## Les listes de Python

- Les listes de Python sont des structures contenant zéro, une ou plusieurs valeurs (pas forcément du même type).
- Une liste se note entre crochets : [ et ]

## Les listes de Python

- Les listes de Python sont des structures contenant zéro, une ou plusieurs valeurs (pas forcément du même type).
- Une liste se note entre crochets : [ et ]
- Les éléments sont séparés par des virgules

## Les listes de Python

- Les listes de Python sont des structures contenant zéro, une ou plusieurs valeurs (pas forcément du même type).
- Une liste se note entre crochets : `[` et `]`
- Les éléments sont séparés par des virgules
- Les éléments d'une liste sont repérés par leur position dans la liste, on dit leur **indice**. Attention, la numérotation commence à zéro.



## Les listes de Python

- Les listes de Python sont des structures contenant zéro, une ou plusieurs valeurs (pas forcément du même type).
- Une liste se note entre crochets : `[` et `]`
- Les éléments sont séparés par des virgules
- Les éléments d'une liste sont repérés par leur position dans la liste, on dit leur **indice**. Attention, la numérotation commence à zéro.

## Les listes de Python

- Les listes de Python sont des structures contenant zéro, une ou plusieurs valeurs (pas forcément du même type).
- Une liste se note entre crochets : `[` et `]`
- Les éléments sont séparés par des virgules
- Les éléments d'une liste sont repérés par leur position dans la liste, on dit leur **indice**. Attention, la numérotation commence à zéro.
- On peut accéder à un élément en indiquant le nom de la liste puis l'indice de cet élément entre crochet

## Les listes de Python

- Les listes de Python sont des structures contenant zéro, une ou plusieurs valeurs (pas forcément du même type).
- Une liste se note entre crochets : `[` et `]`
- Les éléments sont séparés par des virgules
- Les éléments d'une liste sont repérés par leur position dans la liste, on dit leur **indice**. Attention, la numérotation commence à zéro.
- On peut accéder à un élément en indiquant le nom de la liste puis l'indice de cet élément entre crochet
- L'erreur `IndexError` indique qu'on tente d'accéder à un indice qui n'existe pas.

## Les listes de Python

- Les listes de Python sont des structures contenant zéro, une ou plusieurs valeurs (pas forcément du même type).
- Une liste se note entre crochets : `[` et `]`
- Les éléments sont séparés par des virgules
- Les éléments d'une liste sont repérés par leur position dans la liste, on dit leur **indice**. Attention, la numérotation commence à zéro.
- On peut accéder à un élément en indiquant le nom de la liste puis l'indice de cet élément entre crochet
- L'erreur `IndexError` indique qu'on tente d'accéder à un indice qui n'existe pas.
- La longueur d'une liste (ie. son nombre d'éléments) s'obtient à l'aide de la fonction `len`.

## Opérations sur les listes

Les opérations suivantes permettent de manipuler les listes (ajout, suppression, insertion d'éléments). On fera bien attention à la syntaxe on met le nom de la liste suivi d'un point suivi de l'opération à effectuer (voir exemples)

- **append** : permet d'ajouter un élément à la fin d'une liste. Par exemple : `ma_liste.append(elt)` va ajouter `elt` à la fin de `ma_liste`.

## Opérations sur les listes

Les opérations suivantes permettent de manipuler les listes (ajout, suppression, insertion d'éléments). On fera bien attention à la syntaxe on met le nom de la liste suivi d'un point suivi de l'opération à effectuer (voir exemples)

- **append** : permet d'ajouter un élément à la fin d'une liste. Par exemple : `ma_liste.append(elt)` va ajouter `elt` à la fin de `ma_liste`.
- **pop** permet de récupérer un élément de la liste tout en le supprimant de la liste. Par exemple `elt=ma_liste.pop(2)` va mettre dans `elt` `ma_liste[2]` et dans le même temps supprimer cet élément de la liste.

## Opérations sur les listes

Les opérations suivantes permettent de manipuler les listes (ajout, suppression, insertion d'éléments). On fera bien attention à la syntaxe on met le nom de la liste suivi d'un point suivi de l'opération à effectuer (voir exemples)

- **append** : permet d'ajouter un élément à la fin d'une liste. Par exemple : `ma_liste.append(elt)` va ajouter `elt` à la fin de `ma_liste`.
- **pop** permet de récupérer un élément de la liste tout en le supprimant de la liste. Par exemple `elt=ma_liste.pop(2)` va mettre dans `elt` `ma_liste[2]` et dans le même temps supprimer cet élément de la liste.
- **remove** permet de supprimer un élément d'une liste. Par exemple : `ma_liste.remove(elt)` va enlever `elt` de `ma_liste`.

## Opérations sur les listes

Les opérations suivantes permettent de manipuler les listes (ajout, suppression, insertion d'éléments). On fera bien attention à la syntaxe on met le nom de la liste suivi d'un point suivi de l'opération à effectuer (voir exemples)

- **append** : permet d'ajouter un élément à la fin d'une liste. Par exemple : `ma_liste.append(elt)` va ajouter `elt` à la fin de `ma_liste`.
- **pop** permet de récupérer un élément de la liste tout en le supprimant de la liste. Par exemple `elt=ma_liste.pop(2)` va mettre dans `elt` `ma_liste[2]` et dans le même temps supprimer cet élément de la liste.
- **remove** permet de supprimer un élément d'une liste. Par exemple : `ma_liste.remove(elt)` va enlever `elt` de `ma_liste`.
- **insert** permet d'insérer un élément à un indice donnée. Par exemple : `ma_liste.insert(indice,elt)` va insérer `elt` dans `ma_liste` à l'index `indice`.



## Création de listes

On peut créer des listes de diverses façons en Python :

## Création de listes

On peut créer des listes de diverses façons en Python :

- Par ajout **succesif d'élément** on part alors d'une liste (éventuellement vide) et on ajoute chaque élément à l'aide d'instruction **append**.

## Création de listes

On peut créer des listes de diverses façons en Python :

- **Par ajout succesif d'élément** on part alors d'une liste (éventuellement vide) et on ajoute chaque élément à l'aide d'instruction `append`.
- **Par répétition du même élément** on utilise alors le caractère `*` pour indiquer le nombre de répétitions.

## Création de listes

On peut créer des listes de diverses façons en Python :

- **Par ajout succesif d'élément** on part alors d'une liste (éventuellement vide) et on ajoute chaque élément à l'aide d'instruction `append`.
- **Par répétition du même élément** on utilise alors le caractère `*` pour indiquer le nombre de répétitions.

Par exemple : `hesitation = ["euh"]*4`

## Création de listes

On peut créer des listes de diverses façons en Python :

- **Par ajout succesif d'élément** on part alors d'une liste (éventuellement vide) et on ajoute chaque élément à l'aide d'instruction `append`.
- **Par répétition du même élément** on utilise alors le caractère `*` pour indiquer le nombre de répétitions.

Par exemple : `hesitation = ["euh"]*4`

## Création de listes

On peut créer des listes de diverses façons en Python :

- **Par ajout succesif d'élément** on part alors d'une liste (éventuellement vide) et on ajoute chaque élément à l'aide d'instruction `append`.
- **Par répétition du même élément** on utilise alors le caractère `*` pour indiquer le nombre de répétitions.  
Par exemple : `hesitation = ["euh"]*4`
- **Par compréhension**, c'est à dire en indiquant la définition des éléments qui composent la liste.

## Création de listes

On peut créer des listes de diverses façons en Python :

- **Par ajout succesif d'élément** on part alors d'une liste (éventuellement vide) et on ajoute chaque élément à l'aide d'instruction `append`.
- **Par répétition du même élément** on utilise alors le caractère `*` pour indiquer le nombre de répétitions.

Par exemple : `hesitation = ["euh"]*4`

- **Par compréhension**, c'est à dire en indiquant la définition des éléments qui composent la liste.

Par exemple la liste `puissances2 = [1, 2, 4, 8, 16, 32, 64, 128]` est constitué des huit premières puissances de 2

## Création de listes

On peut créer des listes de diverses façons en Python :

- **Par ajout succesif d'élément** on part alors d'une liste (éventuellement vide) et on ajoute chaque élément à l'aide d'instruction `append`.
- **Par répétition du même élément** on utilise alors le caractère `*` pour indiquer le nombre de répétitions.

Par exemple : `hesitation = ["euh"]*4`

- **Par compréhension**, c'est à dire en indiquant la définition des éléments qui composent la liste.

Par exemple la liste `puissances2 = [1, 2, 4, 8, 16, 32, 64, 128]` est constitué des huit premières puissances de 2

Elle contient donc  $2^0, 2^1, 2^2, \dots, 2^7$ , ce qui se traduit en Python par :



## Création de listes

On peut créer des listes de diverses façons en Python :

- **Par ajout succesif d'élément** on part alors d'une liste (éventuellement vide) et on ajoute chaque élément à l'aide d'instruction `append`.
- **Par répétition du même élément** on utilise alors le caractère `*` pour indiquer le nombre de répétitions.

Par exemple : `hesitation = ["euh"]*4`

- **Par compréhension**, c'est à dire en indiquant la définition des éléments qui composent la liste.

Par exemple la liste `puissances2 = [1, 2, 4, 8, 16, 32, 64, 128]` est constitué des huit premières puissances de 2

Elle contient donc  $2^0, 2^1, 2^2, \dots, 2^7$ , ce qui se traduit en Python par :

```
puissances2 = [2**k for k in range(8)]
```

## Exemple

- Crible d'Eratosthène

## Exemple

- Crible d'Eratosthène

## Exemple

- Crible d'Eratosthène

## Exemple

- Crible d'Eratosthène

## Exemple

- Crible d'Eratosthène
- Trie par insertion