

## Création du papier et du crayon

```
1 import turtle
2 papier = turtle.Screen()
3 crayon = turtle.Turtle()
```

## Création du papier et du crayon

```
1 import turtle
2 papier = turtle.Screen()
3 crayon = turtle.Turtle()
```

## Remarques

On peut créer simultanément plusieurs crayons, l'instruction `reset` permet d'effacer la totalité des tracés d'un crayon.

## Propriétés du crayon

## Propriétés du crayon

- `pensize` pour l'épaisseur du trait.

## Propriétés du crayon

- `pensize` pour l'épaisseur du trait.
- `color` pour changer la couleur.

## Propriétés du crayon

- `pensize` pour l'épaisseur du trait.
- `color` pour changer la couleur.
- `penup` et `pendown` pour relever ou abaisser le crayon.

## Propriétés du crayon

- `pensize` pour l'épaisseur du trait.
- `color` pour changer la couleur.
- `penup` et `pendown` pour relever ou abaisser le crayon.
- `hideturtle` et `showturtle` pour masquer ou faire apparaître le crayon.

## Propriétés du crayon

- `pensize` pour l'épaisseur du trait.
- `color` pour changer la couleur.
- `penup` et `pendown` pour relever ou abaisser le crayon.
- `hideturtle` et `showturtle` pour masquer ou faire apparaître le crayon.
- `speed` pour modifier la vitesse de tracé.

## Propriétés du crayon

- `pensize` pour l'épaisseur du trait.
- `color` pour changer la couleur.
- `penup` et `pendown` pour relever ou abaisser le crayon.
- `hideturtle` et `showturtle` pour masquer ou faire apparaître le crayon.
- `speed` pour modifier la vitesse de tracé.

## Exemple

```
1 # Crayon abaissé, rouge, d'épaisseur 3, caché et se déplaç  
   ant à la vitesse maximum  
2 crayon .pendown ()  
3 crayon .pensize (3)  
4 crayon .color ("red")  
5 crayon .hideturtle ()  
6 crayon .speed (10)
```

## Orientation du crayon

## Orientation du crayon

- `setheading` pour fixer l'orientation de la tortue à un angle donnée.

## Orientation du crayon

- `setheading` pour fixer l'orientation de la tortue à un angle donné.
- `left` pour tourner vers la gauche du nombre de degrés donné.

## Orientation du crayon

- `setheading` pour fixer l'orientation de la tortue à un angle donné.
- `left` pour tourner vers la gauche du nombre de degrés donné.
- `right` pour tourner vers la droite du nombre de degrés donné.

## Orientation du crayon

- `setheading` pour fixer l'orientation de la tortue à un angle donné.
- `left` pour tourner vers la gauche du nombre de degrés donné.
- `right` pour tourner vers la droite du nombre de degrés donné.

## Déplacement du crayon

- `goto` pour envoyer la tortue au point de coordonnées  $(x,y)$ .

## Orientation du crayon

- `setheading` pour fixer l'orientation de la tortue à un angle donné.
- `left` pour tourner vers la gauche du nombre de degrés donné.
- `right` pour tourner vers la droite du nombre de degrés donné.

## Déplacement du crayon

- `goto` pour envoyer la tortue au point de coordonnées (x,y).
- `forward` pour faire avancer la tortue de la distance indiquée.

## Orientation du crayon

- `setheading` pour fixer l'orientation de la tortue à un angle donné.
- `left` pour tourner vers la gauche du nombre de degrés donné.
- `right` pour tourner vers la droite du nombre de degrés donné.

## Déplacement du crayon

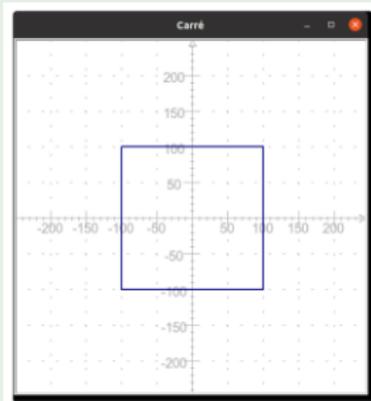
- `goto` pour envoyer la tortue au point de coordonnées  $(x,y)$ .
- `forward` pour faire avancer la tortue de la distance indiquée.
- `backward` pour faire reculer la tortue de la distance indiquée.

## Tracé d'un carré

```
1 crayon . penup ()
2 crayon . setheading (0)
3 crayon . goto (-100,-100)
4 for _ in range (4):
5     crayon . forward (100)
6     crayon . left (90)
```

## Tracé d'un carré

```
1 crayon . penup ()
2 crayon . setheading (0)
3 crayon . goto (-100,-100)
4 for _ in range (4):
5     crayon . forward (100)
6     crayon . left (90)
```



## Programmation en python : généralités

- Python renvoie un message d'erreur lorsqu'il n'arrive pas à interpréter les instructions de votre programme. Prendre l'habitude de **lire attentivement** ces messages, qui sont de **premiers indices** pour déterminer la source de l'erreur

## Programmation en python : généralités

- Python renvoie un message d'erreur lorsqu'il n'arrive pas à interpréter les instructions de votre programme. Prendre l'habitude de **lire attentivement** ces messages, qui sont de **premiers indices** pour déterminer la source de l'erreur
- En Python, les **commentaires** s'écrivent en faisant commencer la ligne par le caractère **#**.

## Programmation en python : généralités

- Python renvoie un message d'erreur lorsqu'il n'arrive pas à interpréter les instructions de votre programme. Prendre l'habitude de **lire attentivement** ces messages, qui sont de **premiers indices** pour déterminer la source de l'erreur
- En Python, les **commentaires** s'écrivent en faisant commencer la ligne par le caractère **#**.
- Le respect de la syntaxe du langage est fondamentale et demande de la rigueur, prendre garde notamment au respect de l'**indentation**.

## Programmation en python : généralités

- Python renvoie un message d'erreur lorsqu'il n'arrive pas à interpréter les instructions de votre programme. Prendre l'habitude de **lire attentivement** ces messages, qui sont de **premiers indices** pour déterminer la source de l'erreur
- En Python, les **commentaires** s'écrivent en faisant commencer la ligne par le caractère **#**.
- Le respect de la syntaxe du langage est fondamentale et demande de la rigueur, prendre garde notamment au respect de l'**indentation**.
- Attention aussi à bien surveiller les correspondances entre les parenthèses mais aussi les guillemets ou les apostrophes qui sont souvent source d'erreurs.

## Instructions conditionnelles

## Instructions conditionnelles

- La syntaxe d'une instruction conditionnelle en Python est :

```
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>.

## Instructions conditionnelles

- La syntaxe d'une instruction conditionnelle en Python est :

```
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>.

- ⚠ On fera bien attention à la syntaxe du langage, et notamment à l'usage du caractère **:** qui suit la condition (et le else) et à l'**indentation**, c'est à dire le décalage des instructions qui doivent s'exécuter.

Exemple de `if ...else`

Compléter le programme suivant afin qu'il affiche "positif" si la variable `r` est supérieure ou égal à 0 et "négatif" sinon.

```
1     .... r >= 0 ..  
2         .... (" positif")  
3     ....:  
4         .....
```

## Exemple de `if ...else`

Compléter le programme suivant afin qu'il affiche "positif" si la variable `r` est supérieure ou égal à 0 et "négatif" sinon.

```
1  if r >= 0:
2      print(" positif")
3  else :
4      print(" négatif")
```

Boucles for

## Boucles for

- Les instructions :

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

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

## Boucles for

- Les instructions :

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

permet de créer 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>
```

permet de créer 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.

## Exemple de for ...range

Quel sera l'affichage produit par le programme suivant ? Expliquer

```
1 for cpt in range(0,5):  
2     print(cpt)
```

## Exemple de for ...range

Quel sera l'affichage produit par le programme suivant ? Expliquer

```
1 for cpt in range(0,5):  
2     print(cpt)
```

Ce programme affiche "0 1 2 3 4".

En effet, la variable `cpt` parcourt les valeurs entières de 0 à 5 mais 5 est exclu. Et à chaque tour de boucle on affiche cette variable grâce à un `print`.

## Définir une fonction en Python

Pour définir une fonction en Python, on utilise la syntaxe suivante :

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

## Définir une fonction en Python

Pour définir une fonction en Python, on utilise la syntaxe suivante :

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

## Exemple

```
1  def plus_grand ( a , b ) :  
2      if a > b :  
3          pg = a  
4      else :  
5          pg = b  
6      return pg
```

## Exercices

- 1 On considère la fonction définie ci-dessous :

```
1     def calcul(x,y):  
2         res = 10*x+y  
3         return res
```

Quel sera la valeur de calcul(2,5)?

## Exercices

- ① On considère la fonction définie ci-dessous :

```
1     def calcul(x,y):  
2         res = 10*x+y  
3         return res
```

Quel sera la valeur de `calcul(2,5)`?

`calcul(2,5)=25`

- ② Ecrire une fonction `est_pair(n)` qui renvoie `True` lorsque l'entier `n` est pair et `False` sinon.

## Correction

```
1 def est_pair(n):
2     ''' Renvoie True ou False suivant que n est pair ou
3         non '''
4     assert type(n) == int, "le paramètre n'est pas un
5         nombre entier"
6     if n%2 == 0:
7         return True
8     else:
9         return False
```

On peut remarquer que c'est la valeur du booléen  $n\%2==0$  qui est renvoyé et donc simplifier l'écriture de cette fonction :

## Correction (meilleure version)

```
1 def est_pair(n):  
2     ''' Renvoie True ou False suivant que n est pair ou  
3         non '''  
4     assert type(n) == int, "le paramètre n'est pas un  
5         nombre entier"  
6     return n%2== 0
```

## Indice d'un élément

- Les éléments d'une liste sont repérés par leur position dans la liste, on dit leur **indice**

## Indice d'un élément

- Les éléments d'une liste sont repérés par leur position dans la liste, on dit leur **indice**
- Attention, la numérotation commence à zéro, l'indice du premier élément de la liste est donc zéro

## Indice d'un élément

- Les éléments d'une liste sont repérés par leur position dans la liste, on dit leur **indice**
- Attention, la numérotation commence à zéro, l'indice du premier élément de la liste est donc zéro
- On peut accéder à un élément en indiquant le nom de la liste puis l'indice de cet élément entre crochet

## Indice d'un élément

- Les éléments d'une liste sont repérés par leur position dans la liste, on dit leur **indice**
- Attention, la numérotation commence à zéro, l'indice du premier élément de la liste est donc 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.



## 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)

- **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)

- **remove** permet de supprimer un élément d'une liste. Par exemple : `ma_liste.remove(elt)` va enlever `elt` de `ma_liste`.
- **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)

- **remove** permet de supprimer un élément d'une liste. Par exemple : `ma_liste.remove(elt)` va enlever `elt` de `ma_liste`.
- **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`.
- **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`.

## 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)

- **remove** permet de supprimer un élément d'une liste. Par exemple : `ma_liste.remove(elt)` va enlever `elt` de `ma_liste`.
- **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`.
- **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`.
- **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.

## 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 pour créer la liste :

```
bavardages = ["bla", "bla", "bla", "bla"]
```

on peut simplement écrire :

## 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 pour créer la liste :

```
bavardages = ["bla", "bla", "bla", "bla"]
```

on peut simplement écrire :

```
bavardages = ["bla"]*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 pour créer la liste :

```
bavardages = ["bla", "bla", "bla", "bla"]
```

on peut simplement écrire :

```
bavardages = ["bla"]*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 pour créer la liste :

```
bavardages = ["bla", "bla", "bla", "bla"]
```

on peut simplement écrire :

```
bavardages = ["bla"]*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 pour créer la liste :

```
bavardages = ["bla", "bla", "bla", "bla"]
```

on peut simplement écrire :

```
bavardages = ["bla"]*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 pour créer la liste :

```
bavardages = ["bla", "bla", "bla", "bla"]
```

on peut simplement écrire :

```
bavardages = ["bla"]*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)]
```