Guide de Programmation Python

Dans ce guide tu apprendras les bases de la programmation python.

Pour débuter ce guide tu auras de la ressource suivante :

Python 3 (Télécharger la dernière version de python), sur le Microsoft Store ou sur le site : https://www.python.org/downloads/

Une fois installée, ouvrez l'application nommée " IDLE (Python X.X XX-bit) "

Cliquez-en haut de la fenêtre, dans les onglets, sur " File ", puis " New File "

Dans la fenêtre qui vient d'ouvrir vous y écrivez vos codes pythons. Ne fermez pas la fenêtre précédente elle vous sert de console (là ou va s'afficher les " print " et donc votre code).

Pour lancer votre code, il est nécessaire de d'abord l'enregistrer dans l'onglet "File" puis "Save" ou Ctrl+S en raccourci clavier.

Pour ouvrir un code qu'on a sauvegarder, on va dans l'onglet "File" puis "Open" et on sélectionne le code.

Pour exécuter le code, allez dans " Run " puis " Run Module ", ou F5 en raccourci.

Le guide comprendra des explications, des codes basiques, des exercices et des idées d'utilisation de ce code à des fins ludiques ou pratiques.

Un conseil pour s'amuser avec Python, c'est d'utiliser ce que vous venez de voir pour un projet personnel, ne visez pas trop grand, par exemple créer votre propre petit jeux, ou un outil de calcul avec ce que vous apprenez, au fur et à mesure que vous découvrez de nouvelles fonctionnalités vous améliorer votre petit projet.

Patrick Nikiel

Affichage et variables

Pour afficher un texte, une information en python, on utilise une fonction nommée " print ". Une fonction est une " action " que fait l'ordinateur, elle se finit obligatoirement par des parenthèses " () ". Certaines fonctions comportent des arguments que l'on met entre parenthèse, un argument est un paramètre qui va changer le fonctionnement de la fonction. Ici le paramètre est " Hello world ", si on a plusieurs paramètres, on les séparera par des virgules.

(Attention a ne pas recopier la numérotation "1. ", elle ne fait pas partie du code, juste une indication du numéro de ligne.)

```
1. print("Hello world")
```

Résultat :

```
Hello world
```

Un des grands principes de l'informatique sont les variables, on assigne une valeur a un nom, que l'on peut appeler plus tard dans le programme (Un programme s'exécute de haut en bas, ligne après ligne).

Exemple :

```
1. username = "steve"
2. print(username)
```

Résultat :

```
steve
```

On définit une variable "username", puis on y assigne la valeur "stev ".

Pour modifier la valeur, il suffit de la redéfinir.
Le nom d'une variable ne doit pas commencer par un chiffre, elle ne peut pas contenir d'espace et il faut eviter d'utiliser des caracteres spéciaux.

Exemple :

```
1. username = "steve"
2. print(username)
3. username = "mario"
4. print(username)
```

Résultat :

```
steve

mario
```

Types de variables

Dans python, il y a plusieurs types de variable, celui qu'on vient d'utiliser est un " string ", il permet de contenir du texte et pour le reconnaitre il commence et se finit par des guillemets.

Voici un tableau des types de variables " primaires " :

Integer int	1 250 -6401	Permet de stocker des nombres entiers
Float float	0.2 5.4 -100.0	Permet de stocker des nombres a virgules
String str	"bonjour" 'Comment va-tu ?' "bonne journée"	Permet de stocker du texte, pour l'ouvrir et le fermer on utilise des guillemets ou des apostrophes.
Boolean bool	True False False	Permet de contenir une valeur qui est soit égale à ¨ Vrai O = True ou ¨ False O = Faux

Il existe des types de variables plus complexes, qui contiennent d'autres variables et qui ont ainsi plusieurs fonctions qui permettent de gérer les variables qu'ils contiennent.

Les types de variables sont les suivants

Liste	[2, 6, 8, 9, 4] ["bonjour", 2, True] ["liste"]	Variable qui contient plusieurs éléments (il est possible de mettre des listes dans des listes)
Dictionnaire	{"username": "Ousama amar", "level": 10, "admin": True}	Un dictionnaire permet de stocker plusieurs valeurs (a droite des deux points), avec chacune une clée (a gauche des deux points).

Operations entre variables

On peut utiliser les variables pour en obtenir des résultats ou les changer à l'aide d'opérations ou de logique. (Une ligne vide dans le code n'est pas comptée, python l'ignore et passe à la suivante)

Pour les opérations entre " Integer " et " Float ", on peut utiliser " + ", " - ", " * " (multiplication) et " / " (division). (On peut aussi faire certaines operations sur des "String")

```
1. banque = 50
2. porte_monnaie = 30
3. depot = 20
4.
5. print(banque)
6. print(porte_monnaie)
7.
8. banque = banque + depot
9. porte_monnaie = porte_monnaie - depot
10.
11.print("Resultat :")
12.print(banque)
13.print(porte_monnaie)
```

```
50
30
Resultat :
50
10
```

On peut utiliser aussi un " + " pour ajouter un texte a un autre, ou un texte dans une variable

```
1. prenom = "Pierre"
2. print("Bonjour " + prenom)
```

```
Bonjour Pierre
```

Conversion de variables

Nous allons essayer d'afficher une variable de type " Integer " (un nombre) avec une variable " String " (un texte)

```
1. habitants = 100
2. print("Il y a : " + habitants + " personnes dans ce village")
```

```
Traceback (most recent call last):
  File "X:/../.py", line 2, in <module>
    print("Il y a : " + habitants + " habitants")
TypeError: can only concatenate str (not "int") to str
```

Nous obtenons une erreur, c'est quelque chose d'assez commun et justement c'est ce qui nous permet de comprendre ce qu'on a de faux et ainsi de progresser. Ignorons la première ligne car cette première ligne n'a pas grand intérêt et ne dépend pas de l'erreur.

Dans la seconde ligne de l'erreur nous avons plus d'information, d'abord nous avez le chemin d'accès de notre code python, là où nous l'avons sauvegardé. Puis on a la ligne de notre code ou l'erreur s'est produite " line 2 "

C'est cette ligne la qui s'affiche dans l'erreur. Puis nous avons l'explication " TypeError ".

Si l'erreur est trop ambiguë il suffit d'écrire le TypeError sur google pour plus de réponse, un bon site qui revient souvent est " StackOverflow ", un forum de programmeurs ou on peut poser des questions et en obtenir des réponses. Ici notre recherche google serait : "can only concatenate str (not "int") to str"

L'erreur nous dit qu'il est impossible de " concaténer " (joindre) un " String " (str) avec un " Integer " (int). Pour faire ça il va falloir convertir le Integer en String

```
1. habitants = 100
2. print("Il y a : " + str(habitants) + " personnes dans ce village")
```

```
Il y a : 100 personnes dans ce village
```

On a utilisé la fonction " str() " qui prend en argument un paramètre que l'on veut convertir en texte peut importe son type de variable. Dans l'exemple suivant nous allons convertir deux fois.

Il existe une fonction par type de variable primaires, dans le tableau en dessous de chaque nom il y a le nom de la fonction.

```
1. distance = 4.3
2. print("Vous avez parcouru : " + str(distance*1000) + " mètres")
3. print("Vous avez parcouru : " + str(int(distance)) + " km")
```

```
Vous avez parcouru : 4300.0 mètres
Vous avez parcouru : 4 km
```

Interactions avec l'utilisateur

Dans python (sans modules), il est possible de faire interagir l'utilisateur en lui demandant la valeur d'une variable. Exemple dans le code suivant :

```
1. print("Entre ton pseudo")
2. username = input()
3. print("Bienvenue " + username)
```

Après l'exécution du code, on peut entrer du texte, ce texte qu'on entre sera stocké dans la variable "username"

```
Entre ton pseudo
LetMeSoloHer
Bienvenue LetMeSoloHer
```

On peut directement ajouter un message au input

```
1. print("Quel est ton métier ?")
2. metier = input("Mon métier est : ")
3. print("Moi aussi je suis " + metier)
```

```
Quel est ton métier ?
Mon métier est : programmeur
Moi aussi je suis programmeur
```

Exercice "Calculatrice 1.0" :

Faire un programme qui prend en entrée deux nombres, puis les multiplies entre eux et l'affiche à l'utilisateur.

Les conditions

Il est possible de vérifier une variable, si elle est égale, inférieur ou supérieur à une autre ou à une constante, la valeur retournée est un booléen. (Pour vérifier si une valeur est égale à une autre, on utilise toujours des doubles signe égal)

```
1. note = 19
2. print(note > 10)
3. print(note == 19)
4. print(note == 18)
```

```
True
True
False
```

On vérifie d'abord si la note est supérieure à 10 (True=Vrai), puis si elle est égale à 19 (True) et on vérifie si elle est égale à 18 (False=Faux)

Un autre exemple :

```
1. print(10.5 > 10.0)
2. print(10.0 < 10.0)
3. print(10.0 >= 10.0)
```

```
True
False
True
```

On peut aussi vérifier si deux String sont égaux :

```
1. name = "Leonard"
2. print(name == "DaVinci")
3. print(name == "Leonard")
4. print(name != "Leonard")
```

```
False
True
False
```

Dans python, on peut "combiner" deux booléen. On peut utiliser "and" pour vérifier si deux booléens sont vrais, si les deux sont vrais ça renvoie vrais, si non, ça retourne faux.

```
1. print(True and False)
2. print(False and False)
3. print(True and True)
```

```
False
False
True
```

Essayons maintenant avec des variables, et non des constantes :

```
1. administrateur = True
2. connecte = False
3. print("Peut utiliser l'application : " + str(administrateur and connecte))
```

```
Peut utiliser l'application :  False
```

L'autre moyen de combiner deux booléens, est d'utiliser "or", si l'un des deux booléens est vrai, alors ça retourne vrais, si aucun des deux n'est vrais, ça retourne faux.

```
1. print(False or False)
2. print(True or False)
3. print(True or True)
```

```
False
True
True
```

De même, avec des variables :

```
1. valeur = 94
2. print("La valeur est entre 90 et 100")
3. print(not (valeur<90 or valeur>100))
```

```
La valeur est entre 90 et 100
True
```

On a utilisé "not", ça donne l'inverse d'un booléen

C'est pratique pour inverser un booléen

```
1. fullscreen = False
2. print("En plein écran : " + str(fullscreen))
3.
4. print("Vous basculer en plein écran")
5. fullscreen = not fullscreen
6. print("En plein écran : " + str(fullscreen))
```

```
En plein écran : False

Vous basculer en plein écran

En plein écran : True
```

Les codes conditionnels

On va maintenant utiliser ce qu'on vient de voir, dans des exemples de codes plus utiles.

On peut décider avec " if " d'exécuter du code si une vérification retourne vrai. Par exemple :

```
1. print("*Vous rentrez dans la salle")
2. metier = "Chef de projet"
3.
4. if metier == "Chef de projet":
5.       print("Bienvenue chef")
6.
7. print("*Vous commencez votre travail")
```

```
*Vous rentrez dans la salle
Bienvenue chef
*Vous commencez votre travail
```

Le code commence par du code comme avant, puis on vérifie si la condition :

"métier == “Chef de projet”"

est vraie, si elle l'est, alors on affiche " Bienvenue chef ", si elle n'était pas vrai, alors on n'afficherait pas le message. (Notez d'ailleurs que tout le code " décalé " est exécuté si la condition est vraie). On appelle un code décalé de 4 espaces (on appuie une fois sur la touche " tab " un code " tabulé ").

Puis à la fin on sort de la condition, donc le message "*Vous commencer votre travail " s'affiche peut importe votre métier.

On peut aussi faire un morceau de code qui s'exécute seulement si la condition n'est pas vérifiée avec " else ".

```
1. print("Vous lancez une pièce")
2. face = False
3. if face == True:
4.       print("La pièce est tombée sur face")
5. else:
6.       print("La pièce est tombée sur pile")
7. print("Vous faite tomber la pièce")
```

```
Vous lancez une pièce
La pièce est tombée sur pile
Vous faite tomber la pièce
```

Pour ajouter plusieurs conditions avec un " else ", on peut utiliser " elif "

```
1. print("Vous lancez un dé a 4 faces")
2. de = 2
3. if de == 1:
4.       print("Le dé est tombé sur 1")
5. elif de == 2:
6.       print("Le dé est tombé sur 2")
7. elif de == 3:
8.       print("Le dé est tombé sur 3")
9. else:
10.      print("Le dé est tombé sur 4")
```

```
Le dé est tombé sur 2
```

Dans cet exemple, même si la variable " de " n'est pas entre 1 et 3, elle exécute la dernière condition qui dit que le dé est tombé sur 4.

```
1. print("Vous lancez un dé a 4 faces")
2. de = 11
3. if de == 1:
4.       print("Le dé est tombé sur 1")
5. elif de == 2:
6.       print("Le dé est tombé sur 2")
7. elif de == 3:
8.       print("Le dé est tombé sur 3")
9. else:
10.      print("Le dé est tombé sur 4")
```

```
Le dé est tombé sur 4
```

Exercice "Calculatrice 2.0" :

Demandez à l'utilisateur quel type d'opération (addition, soustraction, multiplication, , division) il veut faire, puis demander lui deux nombres avec lequel il veut faire l'opération. Renvoyez lui le résultat.

Les boucles à condition

Il y a possibilité de répéter un morceau de code X nombre de fois, ou le répéter jusqu'à ce qu'une condition soit vraie. Commençons par le répéter jusqu'à ce qu'une condition soit vraie, grâce à une boucle "while".
Pour qu'une boucle "while" se termine, il faut qu'a un moment sa condition retourne "Faux" (elle continue d'exécuter le code tant que sa condition est vraie)

```
1. compteur = 0
2. while compteur < 5:
3.         compteur = compteur + 1
4.         print(compteur)
```

```
1
2
3
4
5
```

Par exemple une boucle avec une condition toujours vraie ne s'arrêteras jamais, par exemple :

```
1. while 1==1:
2.         print("C'est une boucle infinie !")
```

```
C'est une boucle infinie !
C'est une boucle infinie !
C'est une boucle infinie !
…
```

Et une boucle avec une condition fausse avant son exécution ne s'exécutera pas :

```
1. print("La boucle ne marchera pas")
2. while "burger"=="salade":
3.         print("Je suis la boucle")
4. print("Fin du code")
```

```
La boucle ne marchera pas
Fin du code
```

Exercice "Compte à rebours" :

Faites un compte à rebours, qui compte de 10 à 0.

Les listes

Parmi les variables "complexes", il y a les listes, une liste se présente comme ceci :

```
1. une_liste_vide = []
2. une_liste_pleine = [False, 1, "deux", 3.0]
```

Une liste, comme au-dessus, peut contenir plusieurs types d'éléments différents, ou un seul type, elle n'a pas de taille limite.

Les listes possèdent plusieurs méthodes, par exemple pour ajouter un élément dans une liste après l'avoir créé, on utilise "append(arg)"

```
1. inventaire = ["lance", "bouclier", "casque"]
2. print("Votre inventaire : " + str(inventaire))
3.
4. print("Quel objet avez-vous ramassé ?")
5. objet = input()
6.
7. inventaire.append(objet)
8. print("Votre inventaire : " + str(inventaire))
```

```
Votre inventaire : ['lance', 'bouclier', 'casque']
Quel objet avez-vous ramassé ?
gateau
Votre inventaire : ['lance', 'bouclier', 'casque', 'gateau']
```

Pour récupérer la valeur d'un élément dans une liste, on utilise sa position. Dans python, les positions de listes commence à 0 et pas 1.

```
1. recette = ["sel", "oeuf", "sucre", "farine"]
2. premier_ingredient = recette[0]
3. print("Le premier ingredient est : " + premier_ingredient)
4. print("Le deuxieme ingredient est : " + recette[1])
```

```
Le premier ingredient est : sel
Le deuxieme ingredient est : oeuf
```

Une fonction globale qu'on peut utiliser est "len(arg)". Elle donne la longueur d'une variable.
Par exemple on peut obtenir la longueur d'une liste.

```
1. livres = ["Gorgias", "Berserk", "L'Étranger", "Steel Ball Run"]
2. nombre_de_livres = len(livres)
3. print("Vous avez : " + str(nombre_de_livres) + " livres")
```

```
Vous avez : 4 livres
```

On peut utiliser cette fonction pour par exemple obtenir le dernier élément d'une liste, peu importe sa taille (comme les positions commence à 0 en python, on fera la longueur – 1)

```
1. achats = ["Lampe", "Table", "Armoire", "Bureau"]
2. print("Votre dernier achat est : " + achats[len(achats) - 1])
3. print("Votre avant dernier achat est : " + achats[len(achats) - 2])
```

```
Votre dernier achat est : Bureau
Votre avant dernier achat est : Armoire
```

Pour retirer un élément d'une liste on utilise sa position, avec la fonction "pop(arg)"

```
1. contacts = ["Patron", "Collegue", "Secretaire"]
2. print("Vos contacts sont : " + str(contacts))
3. print("Suppression du contact 'Collegue'")
4. contacts.pop(1)
5. print("Vos contacts sont : " + str(contacts))
```

```
Vos contacts sont : ['Patron', 'Collegue', 'Secretaire']
Suppression du contact 'Collegue'
Vos contacts sont : ['Patron', 'Secretaire']
```

On peut aussi vérifier si un élément est dans une liste avec "in"

```
1. backpack = ["clef", "livre"]
2. print("Vous trouvez une porte")
3. if "clef" in backpack:
4.       print("Vous ouvrez la porte")
```

```
Vous trouvez une porte
Vous ouvrez la porte
```

Les boucles à listes

On a vu précédemment des boucles à condition, on va maintenant aborder des boucles qui parcourent pour chaque élément d'une liste, un code.
On utilise "for element in my_list:"

Dans la syntaxe, "element" est une variable, qu'on pourra utiliser dans le code tabulé, elle sera chaque element de la liste "my_liste" (dans cet exemple de syntaxe), un exemple d'utilisation :

```
1. equipement = ["Casque", "Fusil", "Gilet"]
2. print("Votre equipement : ")
3. for objet in equipement:
4.       print("-" + objet)
```

```
Votre equipement :
-Casque
-Fusil
-Gilet
```

On peut aussi générer avec python des listes (un sous-type de liste, qu'on peut convertir en liste avec la fonction globale "list(arg)"), qui contiennent les numéros d'un nombre A à un nombre B, avec la fonction globale "range(A,B)"

Par exemple on va d'abord afficher "range" sous force de liste (car c'est un sous type de liste), pour voir à quoi ça ressemble

```
1. liste_range = range(0, 10)
2. print(list(liste_range))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

On peut y mettre un pas, par exemple afficher toutes les deux valeurs avec le dernier argument, ici 2 :

```
1. liste_range = range(0, 10, 2)
2. print(list(liste_range))
```

```
[0, 2, 4, 6, 8]
```

On peut donc l'utiliser dans une boucle (qu'elle soit convertie ou pas en liste, ça donne le même résultat), ça affiche tout les éléments de la liste, un par un (attention, ce ne sont pas les numéros des lignes dans la console, mais la variable "number" à chaque exécution du code tabulé) :

```
1. for number in range(0, 7):
2.         print(number)
```

```
0
1
2
3
4
5
6
```

On peut reproduire le premier exemple de ce chapitre avec "range"

```
1. equipement = ["Casque", "Fusil", "Gilet"]
2. print("Votre equipement : ")
3. for position in range(0,len(equipement)):
4.         print("-" + equipement[position])
```

```
Votre equipement :
-Casque
-Fusil
-Gilet
```

On obtient le même résultat que précédemment, la différence est que maintenant on a dans notre boucle, la position de l'objet en même temp que l'objet

```
1. equipement = ["Casque", "Fusil", "Gilet"]
2. print("Votre equipement : ")
3. for position in range(0,len(equipement)):
4.         print(str(position) + " - " + equipement[position])
```

```
Votre equipement :
0 - Casque
1 - Fusil
2 - Gilet
```

Et si on veut commencer les indexes à partir de 1 :

```
1. equipement = ["Casque", "Fusil", "Gilet"]
2. print("Votre equipement : ")
3. for position in range(0,len(equipement)):
4.       print(str(position + 1) + " - " + equipement[position])
```

```
Votre equipement :
1 - Casque
2 - Fusil
3 - Gilet
```

Lisibilité et modularité du code

Pour que le code soit plus lisible, on peut utiliser des commentaires, les commentaires commencent toujours par des #. Il ne faut pas négliger les commentaires car ça peut s'avérer utile quand on revient sur ancien projet, ou si on veut donner des explications a d'autre personnes qui lisent notre code.

```
1. #print("Ceci n'est pas exécutable")
2. #Nous allons afficher un message
3. print("Ceci est exécuté")
```

```
Ceci est du exécuté
```

On peut aussi mettre des commentaires sur plusieurs lignes de la façon suivante :

```
1. """Ceci est un
2. commentaire
3. sur
4. plusieurs
5. lignes"""
6.
7. print("Ceci est du code")
```

```
Ceci est du code
```

Il arrive parfois qu'on a un morceau de code, que l'on veut répéter, pour éviter de le recopier plusieurs fois, on peut créer notre propre fonction avec "def"

Attention, on ne peut qu'utiliser notre fonction après l'avoir créée

Et pour utiliser une variable qui est créée avant notre fonction, on doit préciser qu'on utilise une variable "globale", qu'on peut accéder partout dans notre code :

```
1. vie = 100
2. #On créer une fonction qui fait perdre 10 points de vie
3. def prendre_degats():
4.       global vie
5.       vie = vie - 10
6.       print("Vous avez " + str(vie) + " points de vie")
7.
8. print("Le goblin attaque !")
9. prendre_degats()
10.print("Le loup attaque !")
11.prendre_degats()
12.print("Le squelette attaque !")
13.prendre_degats()
14.print("Fin du combat !")
```

```
Le goblin attaque !
Vous avez 90 points de vie
Le loup attaque !
Vous avez 80 points de vie
Le squelette attaque !
Vous avez 70 points de vie
Fin du combat !
```

On peut aussi définir, dans nos fonctions, un argument, par exemple :

```
1. #On créer la fonction dire_bienvenue
2. #Cette fonction affiche "Bienvenue" + le nom qu'on met en paramètre
3.
4. def dire_bienvenue(prenom):
5.       print("Bienvenue " + prenom)
6.
7. #On utilise la fonction
8. dire_bienvenue("Walter")
9. dire_bienvenue("Jesse")
```

```
Bienvenue Walter
Bienvenue Jesse
```

On peut, en définir plusieurs

```
1. #On créer la fonction "creer_compte", qui prend en argument le nom du client
2. #et le nombre de sous qu'il depose a la creation du compte
3. banque_argent = 0
4. banque_clients = []
5. def creer_compte(nom, depot):
6.       global banque_argent
7.       global banque_clients
8.       banque_argent += depot
9.       banque_clients.append(nom)
10.
11.#On créer des comptes clients
12.creer_compte("Salamanca", 50000)
13.creer_compte("Fring", 20500)
14.#Affiche le résultat
15.print(banque_argent)
16.print(banque_clients)
```

```
70500
['Salamanca', 'Fring']
```

On peut aussi utiliser "return" pour qu'une fonction nous retourne une valeur, tout comme une variable

```
1. porte_monnaie = 50
2. soldes = False
3.
4. #On défini une fonction "prix_vetement" qui renvoie le prix d'un vetement
5. #La fonction renvoie la moitiée du prix du vétement s'il y a des soldes
6. def prix_vetement(prix):
7.     global soldes
8.     global porte_monnaie
9.     if soldes:
10.        return int(float(prix) / 2)
11.    else:
12.        return prix
13.
14.#Les soldes sont désactivée par défaut
15.print("Vous allez en magasin avec " + str(porte_monnaie) + "€.")
16.print("Vous regarder une veste, elle coute : " + str(prix_vetement(20)))
17.print("Si vous achetez la veste il vous restera : ")
18.print(str(porte_monnaie-prix_vetement(20)) + "€.")
19.
20.print("C'est le Black Friday, vous avez déchiré votre veste")
21.soldes = True
22.
23.#Les soldes sont maintenant active, la fonction renvoie une autre valeur
24.print("Vous regardez le prix de la même veste : " + str(prix_vetement(20)))
25.print("Si vous achetez la veste il vous restera : ")
26.print(str(porte_monnaie-prix_vetement(20)) + "€.")
```

```
Vous allez en magasin avec 50€.
Vous regarder une veste, elle coute : 20
Si vous achetez la veste il vous restera :
30€.
C'est le Black Friday, vous avez déchiré votre veste
Vous regardez le prix de la même veste : 10
Si vous achetez la veste il vous restera :
40€.
```

Lorsqu'une fonction retourne une valeur, il est important de noter que la fonction se finit et n'exécute pas le code qui suit.
Certains langages de programmations demandent obligatoirement à chaque fonction une valeur à retourner. On peut dans ce cas, retourner rien

```
1. cache = True
2. print("Pouvez-vous le voir ?")
3. def message():
4.     if cache:
5.         return
6.     print("C'est un message secret !")
7. message()
```

```
Pouvez-vous le voir ?
```