

Cours 12:

Importation des modules et Debugger

Présentation par Tommy Gagnon Joyal



Table des matières

01

Rappel et présences

Rappel et prise des
présences

02

Rappel des fonctions

Rappel sur l'explication
sur les fonctions

03

Utilisation des Import

Qu'est-ce qu'un import ?

04

Le Debugger

Comment utiliser le
debugger

05

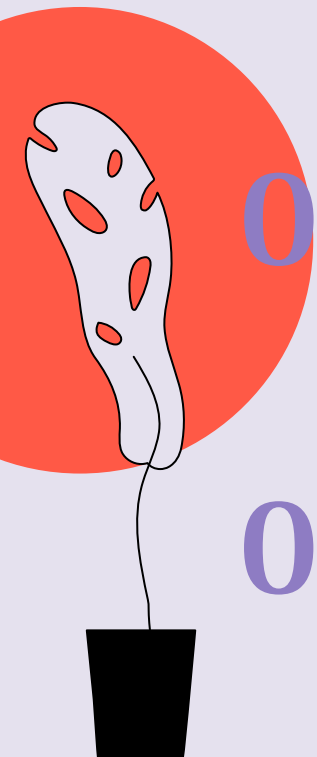
Pause

Café 🥤 😊

06

Exercices en classe

...





01

Rappel



Prochaines dates

- Quiz #4: cours #13 (prochain cours)
- Quiz Bonus: cours #14



Prochaines dates

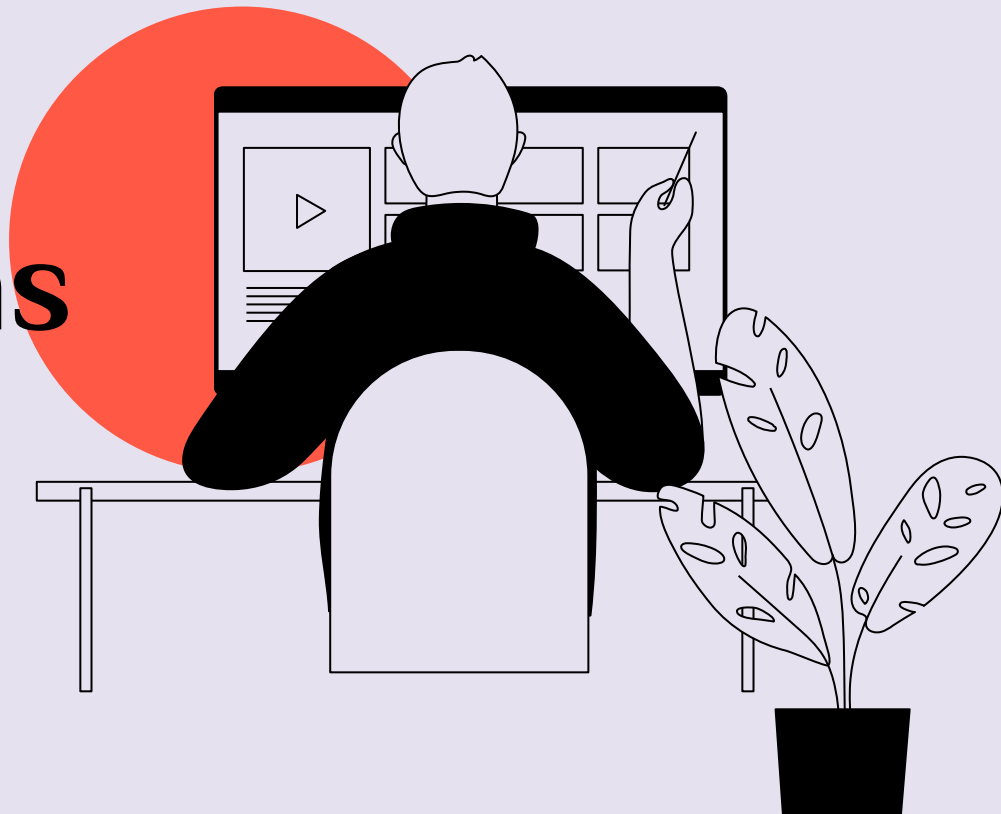


- **TP2: À donner Aujourd'hui**

02

Les fonctions

Rappel sur l'explication sur les
fonctions



Les fonctions

Rappel

- Une fonction est un bloc de code qui s'exécute uniquement lorsqu'elle est appelée.
- Vous pouvez transmettre des données, appelées paramètres, dans une fonction.
- Une fonction peut renvoyer des données en tant que résultat.



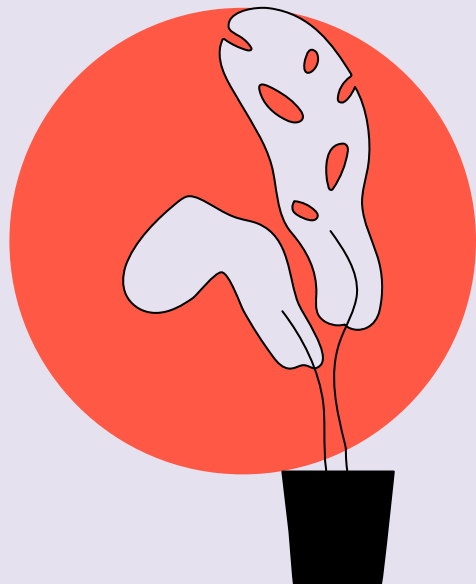
Les fonctions

Rappel

Création d'une fonction:

- En Python, une fonction est définie en utilisant le mot-clé def :

```
def my_function():  
    print("Hello from a function")
```



Les fonctions

Rappel

Appeler une fonction:

- Pour appeler une fonction, utilisez le nom de la fonction suivi de parenthèses :

```
def my_function():  
    print("Hello from a function")
```

```
my_function()
```



Les fonctions

Rappel

Arguments:

- Des informations peuvent être transmises aux fonctions sous forme d'arguments.
- Les arguments sont spécifiés après le nom de la fonction, à l'intérieur des parenthèses. Vous pouvez ajouter autant d'arguments que vous le souhaitez, il suffit de les séparer par une virgule.



Les fonctions

Rappel

Arguments:

- L'exemple suivant comporte une fonction avec un argument (fname). Lorsque la fonction est appelée, nous transmettons un prénom, qui est utilisé à l'intérieur de la fonction pour afficher le nom complet :

```
def my_function(fname):  
    print(fname + " Refsnes")  
  
my_function("Emil")  
my_function("Tobias")  
my_function("Linus")
```



Les fonctions

Rappel

Valeur par défaut des paramètres:

- L'exemple suivant montre comment utiliser une valeur par défaut pour un paramètre.
- Si nous appelons la fonction sans argument, elle utilise la valeur par défaut :

```
def my_function(country = "Norway"):  
    print("I am from " + country)
```

```
my_function("Sweden")  
my_function("India")  
my_function()  
my_function("Brazil")
```

Les fonctions

Rappel

Valeurs de retour:

- Pour permettre à une fonction de renvoyer une valeur, utilisez l'instruction return :

```
def my_function(x):  
    return 5 * x  
  
print(my_function(3))  
print(my_function(5))  
print(my_function(9))
```



Les fonctions

Rappel

Et plus encore !

- https://www.w3schools.com/python/python_functions.asp





03

Utilisation des Import

Qu'est-ce qu'un module ?

Considérez un module comme une bibliothèque de code.

Un fichier contenant un ensemble de fonctions que vous souhaitez inclure dans votre application.



Création d'un module



Pour créer un module, il suffit de générer un nouveau fichier et d'y déposer les fonctions que l'on souhaite utiliser.

Créons par exemple un fichier appelé **mymodule.py** puis on ajoute les fonctions désiré a l'interieur.

```
def greeting(name):  
    print("Hello, " + name)
```

Utiliser un module

Maintenant, nous pouvons utiliser le module que nous venons de créer en utilisant l'instruction **import** :

```
import mymodule
```

```
mymodule.greeting("Jonathan")
```



Les variables dans un module

Le module peut contenir des fonctions, comme déjà décrit, mais aussi des variables de tous types (tableaux, dictionnaires, objets, etc.):

```
person1 = {  
    "name": "John",  
    "age": 36,  
    "country": "Norway"  
}
```

```
import mymodule
```

```
a = mymodule.person1["age"]  
print(a)
```



Nommer un module

Vous pouvez nommer le fichier du module comme vous le souhaitez, mais il doit avoir l'extension de fichier .py.

Rebaptiser un module

Vous pouvez créer un alias lorsque vous importez un module, en utilisant le mot-clé **as**:

```
import mymodule as mx

a = mx.person1["age"]
print(a)
```



Importer à partir d'un module

Vous pouvez choisir d'importer seulement certaines parties d'un module en utilisant le mot-clé **from**.

```
def greeting(name):  
    print("Hello, " + name)  
  
person1 = {  
    "name": "John",  
    "age": 36,  
    "country": "Norway"  
}
```

```
from mymodule import person1  
  
print (person1["age"])
```



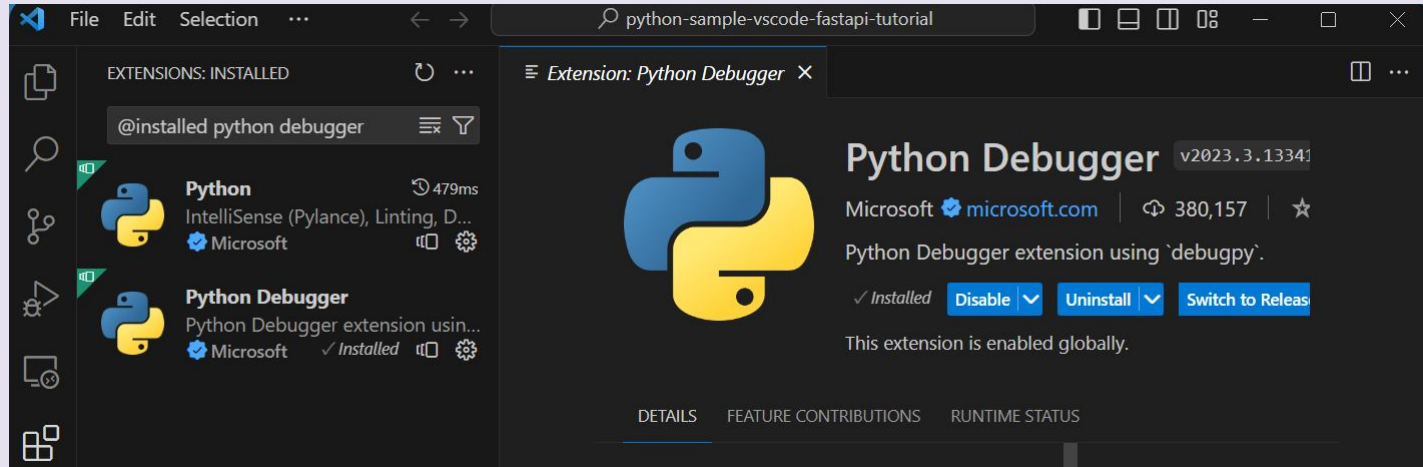
04

Le Debugger



Extension Débogueur Python

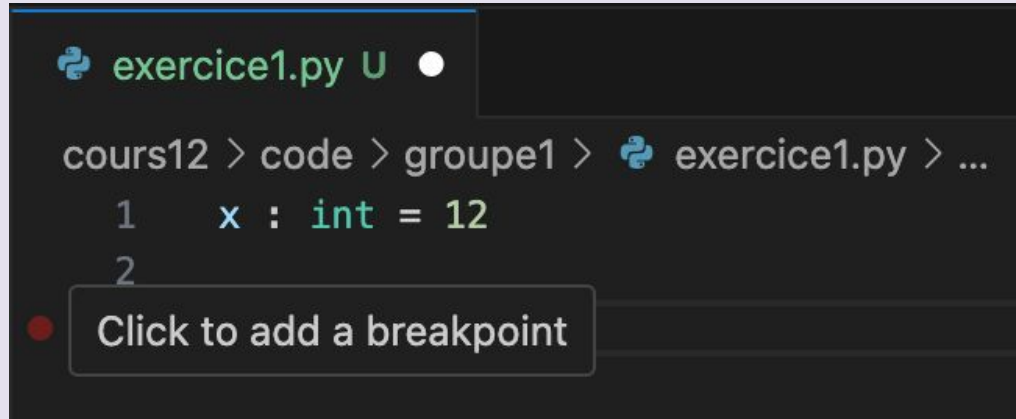
L'extension Débogueur Python est automatiquement installée avec l'extension Python pour VS Code. Elle offre des fonctionnalités de débogage avec debugpy pour plusieurs types d'applications Python, y compris les scripts, les applications web, les processus à distance et plus encore.



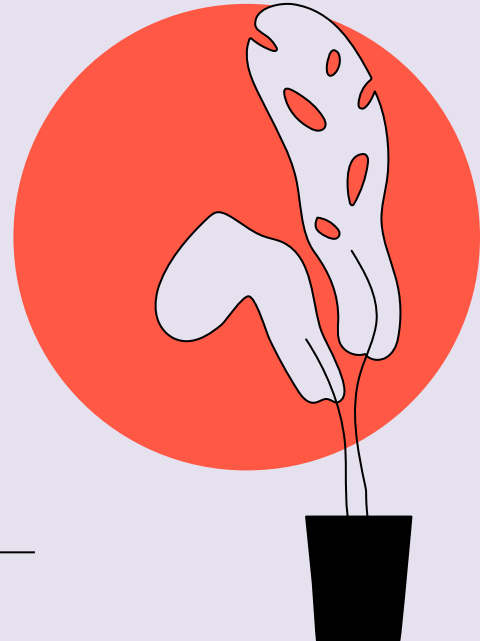


Les points d'arrêts! (Breakpoints)

Un breakpoint dans VS Code est un point d'arrêt défini par l'utilisateur dans le code source lorsqu'il souhaite arrêter l'exécution du programme à un point spécifique pour inspecter son état.



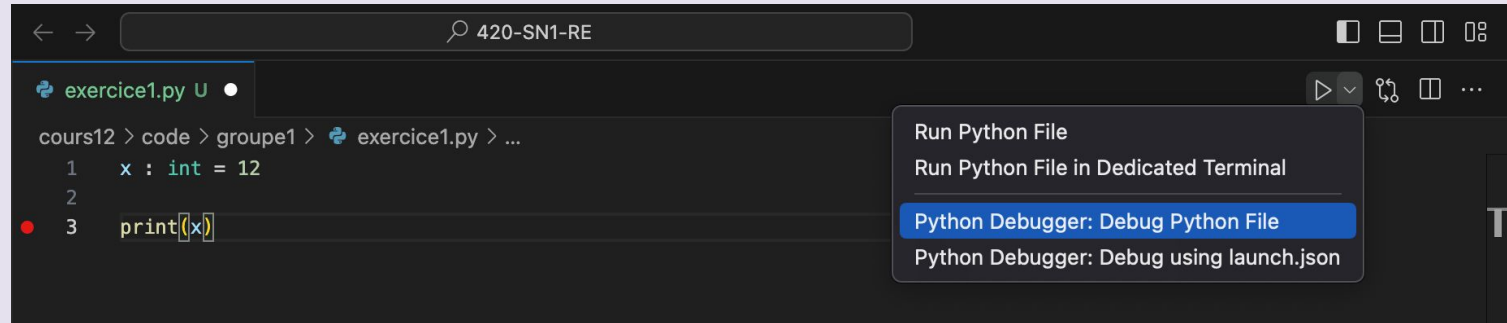
```
exercice1.py U ●  
cours12 > code > groupe1 > exercice1.py > ...  
1 x : int = 12  
2  
● Click to add a breakpoint
```



Une fois les breakpoints placés aux endroits désiré, il suffit de démarrer le programmer en mode debug. Pour cela, il existe deux façons.

1. Démarrer le programme depuis le bouton “play”, en prenant la peine de sélectionner l’option “python debugger”
2. Démarrer le programme depuis l’onglet “Run and debug” situé à gauche de l’écran

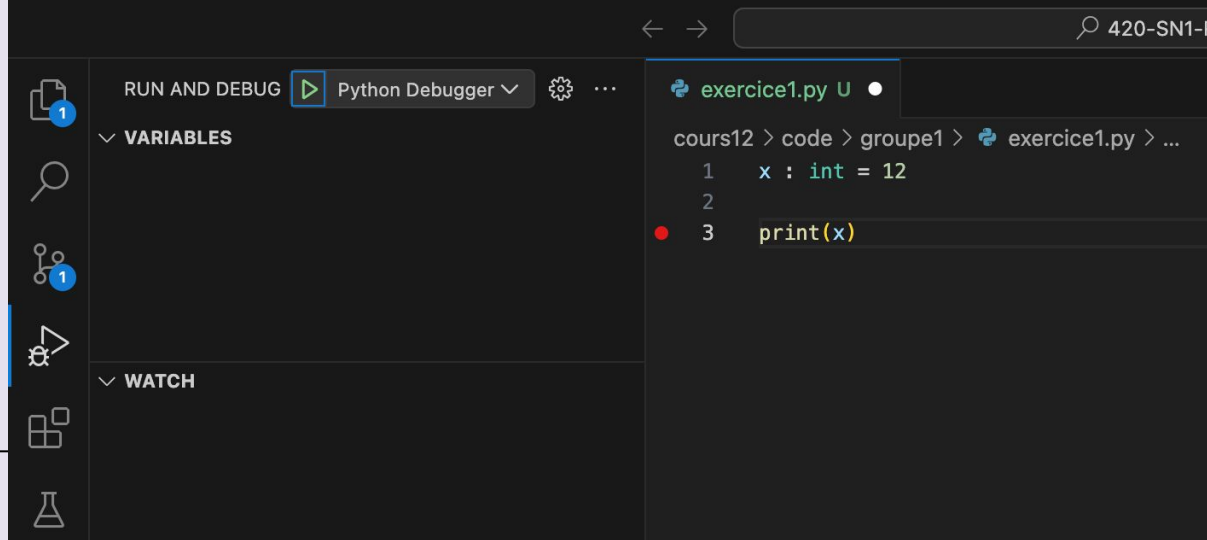
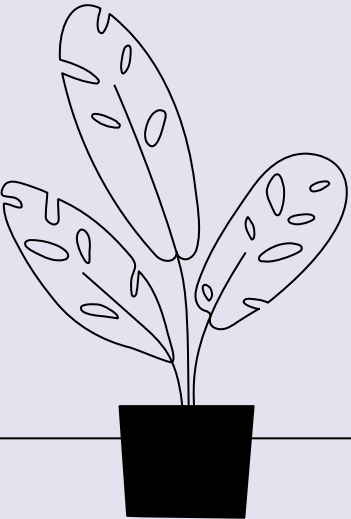
Option 1 :



Une fois les breakpoints placés aux endroits désiré, il suffit de démarrer le programmer en mode debug. Pour cela, il existe deux façons.

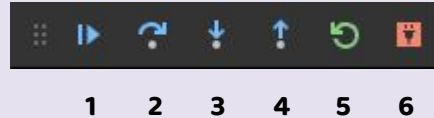
1. Démarrer le programme depuis le bouton "play", en prenant la peine de sélectionner l'option "python debugger"
2. **Démarrer le programme depuis l'onglet "Run and debug" situé à gauche de l'écran**

Option 2 :

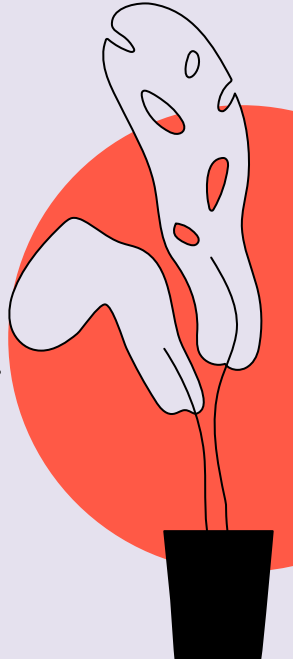


La barre de navigation du debugger

Une fois les breakpoints placés aux endroits désiré et démarré le programme en mode debug, il suffit de naviguer avec la barre de navigation du debugger.



1. Continue : Permet simplement de continuer le flow naturel du programme une fois un point d'arrêt atteint.
2. Step Over: Permet de passer à la ligne suivant, même si cette ligne n'a pas de breakpoint.
3. Step Into: Permet d'entrer plus en détail dans l'action effectué dans le code. Si par exemple, la ligne sélectionnée appel une fonction, ce bouton permettra de voir qu'est-ce qu'il y a l'intérieur de la fonction.
4. Step Out: Permet de sortir de d'une fonction. Il s'agit de l'action opposé du bouton Step Into.

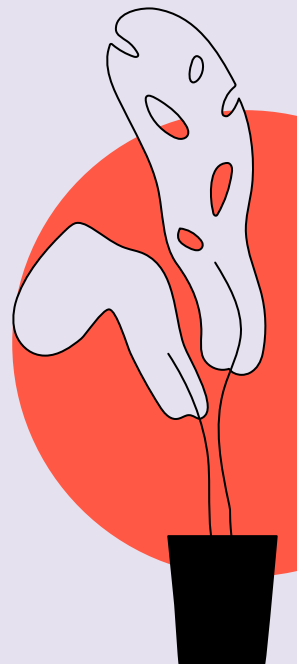


La barre de navigation du debugger

Une fois les breakpoints placés aux endroits désiré et démarré le programme en mode debug, il suffit de naviguer avec la barre de navigation du debugger.



1. Continue : Permet simplement de continuer le flow naturel du programme une fois un point d'arrêt atteint.
2. Step Over: Permet de passer à la ligne suivant, même si cette ligne n'a pas de breakpoint.
3. Step Into: Permet d'entrer plus en détail dans l'action effectué dans le code. Si par exemple, la ligne sélectionnée appel une fonction, ce bouton permettra de voir qu'est-ce qu'il y a l'intérieur de la fonction.
4. Step Out: Permet de sortir de d'une fonction. Il s'agit de l'action opposé du bouton Step Into.
5. Restart: Permet de redémarrer le programme, toujours en mode Debug.
6. Stop: Permet de fermer complètement l'application qui roule présentement. L'on peut donc fermer l'application rapidement.



Resources

Langage:

- Interprétée (Comme Python): Exécuté ligne par ligne.
- Compilée: Le code est complètement transformé en Assembleur pour être exécuté.

Programmation:

- Procédurale: Une suite d'instructions (souvent réunies en fonctions) exécutées par une machine.

- Orientée objet (POO) (Comme Python): Chaque programme est constitué d'entités appelées objets, qui ne sont pas facilement accessibles et modifiables.

Type de variable:

- Variable: Symboles qui associent un nom (l'identifiant) à une valeur.
 - CONSTANTE: Une variable qui ne devrait pas changer pendant l'exécution d'un programme.
-

Resources

- Variable Booléenne: Relatif à l'algèbre de Boole. Qui ne peut prendre que deux valeurs distinctes.
- Commentaire: Permet de documenter du code dans un fichier. Celui-ci ne sera pas lu par le langage en question.
- Faiblement typé: Se préoccupe peu des types. Cela permet de chaîner des valeurs de différents types.
- Typé dynamiquement (Comme Python): Il va définir lui-même le type selon la nature de la valeur .
- Orientée objet (POO) (Comme Python): Chaque programme est constitué d'entités appelées objets, qui ne sont pas facilement accessibles et modifiables.

Typage:

- Fortement typé : est un langage dans lequel les types utilisés dans le code source (fonction, variable, etc.) sont vérifiés au moment de la compilation.
-

Resources

Énoncés conditionnels:

- L'énoncé 'if' et ses variantes: Exécute un bloc de code si une condition évalue à true.

- La boucle "while" est également une structure de répétition. Elle permet l'exécution répétée des actions définies à l'intérieur, en utilisant l'indentation pour délimiter le bloc de code à répéter.

Énoncés conditionnels:

- La boucle "for" est une structure de répétition. Elle permet l'exécution répétée des actions définies à l'intérieur, en utilisant l'indentation pour délimiter le bloc de code à répéter.
-