

2N6 Programmation 2



pythonTM

Rencontre 2 – Listes dans Python



Les listes

Les possibilités qu'elles offrent



Les listes sont des structures de données qui permettent de stocker plusieurs objets dans une même variable.

Les listes sont des structures ordonnées. Les objets stockés dans la liste ont un ordre et sont accessibles par leur indice / position qu'on nomme souvent index.

```
liste = ['Ford Mustang 1964', 'Reliant Robin 1988', 'Toyota Tercel 1991']
```

```
print(liste)
# ['Ford Mustang 1964', 'Reliant Robin 1988', 'Toyota Tercel 1991']
print(liste[1])
# 'Reliant Robin 1988'
print(liste[2])
# 'Toyota Tercel 1991'
```

Accéder aux méthodes

On lance
l'interpréteur Python

Comme pour toute
autre classe :

« dir() » nous donnent
la liste des méthodes
disponibles

Méthodes
spéciales
(dunder)

Méthodes des
objets « list »

```
Invite de commandes - python

C:\Users\pierre-paul.gallant>python
Python 3.10.5 (tags/v3.10.5:f377153, Jun  6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> dir(list)
['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>>
```



Méthodes les plus courantes :

- > **list.append(objet)**: ajoute un objet à la fin de la liste
- > **list.clear()**: Vide une liste
- > **list.count()** -> **int** : Compte le nombre de fois qu'une valeur apparaît dans la liste
- > **list.extend([objet1, objet2])**: Ajoute les valeurs d'une liste à la liste existante (fonctionne différemment de append())
- > **list.index(objet)**: Retourne l'index (i.e; la position) de la valeur passée
- > **list.insert(index, objet)** : Ajoute un objet à la position indiqué
- > **list.remove(objet)** : retire le premier objet correspondant à la valeur passée
- > **list.pop(index)** -> **objet**: retire ET retourne l'objet à l'index indiqué. Par défaut : le dernier objet
- > **list.reverse()** : inverse l'ordre des objets dans la liste
- > **list.sort()** : trie la liste en ordre croissant
- > **list.copie()** -> **list** : Crée un nouvel objet qui est une copie « peu profonde » de la liste

Listes (méthodes) - explication

> `append()`: ajoute un objet à la fin de la liste

```
>>> cours = ["prog 1", "reseau 1"]
>>> cours.append("prog 2")
>>> print(cours)
['prog 1', 'reseau 1', 'prog 2']
>>>
```

> `clear()`: Vide une liste

```
>>> cours.clear()
>>> print(cours)
[]
```

> `count()`: Compte le nombre de fois qu'une valeur apparaît dans la liste

```
>>> cours.count("prog 1")
3
>>> cours = ["prog 1", "res 1", "prog 2", "prog 1"]
>>> cours.count("prog 1")
2
```

Listes (méthodes) - explication

- > `extend()`: Ajoute les valeurs d'une liste à la liste existante (fonctionne différemment de `append()`)
- > `index()`: Retourne l'index (i.e; la position) de la valeur passée
- > `insert()` : Ajoute un objet à la position indiqué

```
>>> cours = ["prog 1", "res 1", "prog 2", "prog 1"]
>>> cours2 = ["philo", "anglais", "français"]
>>> cours.extend(cours2)
>>> print(cours)
['prog 1', 'res 1', 'prog 2', 'prog 1', 'philo', 'anglais', 'français']
>>> _
```

```
>>> print(cours.index("res 1"))
1
>>>
```

```
>>> print(cours)
['prog 1', 'res 1', 'prog 2', 'prog 1', 'philo', 'anglais', 'français']
>>> cours.insert(2, "prog 3")
>>> print(cours)
['prog 1', 'res 1', 'prog 3', 'prog 2', 'prog 1', 'philo', 'anglais', 'français']
>>> _
```

Listes (méthodes) - explication

- > `remove()` : retire le premier objet correspondant à la valeur passée
- > `pop()` : retire ET retourne l'objet à l'index indiqué. Par défaut : le dernier objet
- > `reverse()` : inverse l'ordre des objets dans la liste
- > `sort()` : trie la liste en ordre croissant

```
>>> cours.remove("prog 1")
>>> print(cours)
['res 1', 'prog 3', 'prog 2', 'prog 1', 'philos', 'anglais', 'français']
>>>
```

```
>>> cours_retirer = cours.pop(1)
>>> print(cours)
['res 1', 'prog 2', 'prog 1', 'philos', 'anglais', 'français']
>>> print(cours_retirer)
prog 3
>>> _
```

```
>>> cours.reverse()
>>> print(cours)
['français', 'anglais', 'philos', 'prog 1', 'prog 2', 'res 1']
>>>
```

```
>>> cours.sort()
>>> print(cours)
['anglais', 'français', 'philos', 'prog 1', 'prog 2', 'res 1']
>>> _
```


Listes (méthode copy)

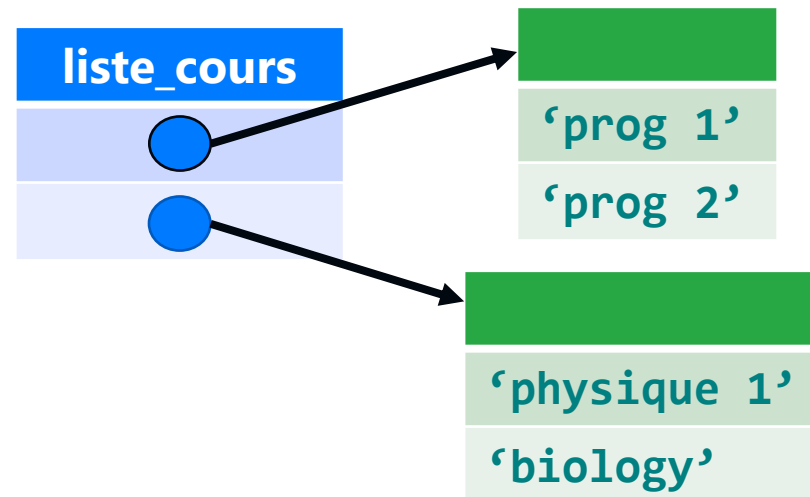


> `copy()` : retourne une copie « peu profonde » de la liste.

Génère une nouvelle liste contenant les mêmes références aux mêmes objets que la première liste.

```
liste_cours = [['prog 1', 'prog 2'], ['physique 1', 'biology']]
```

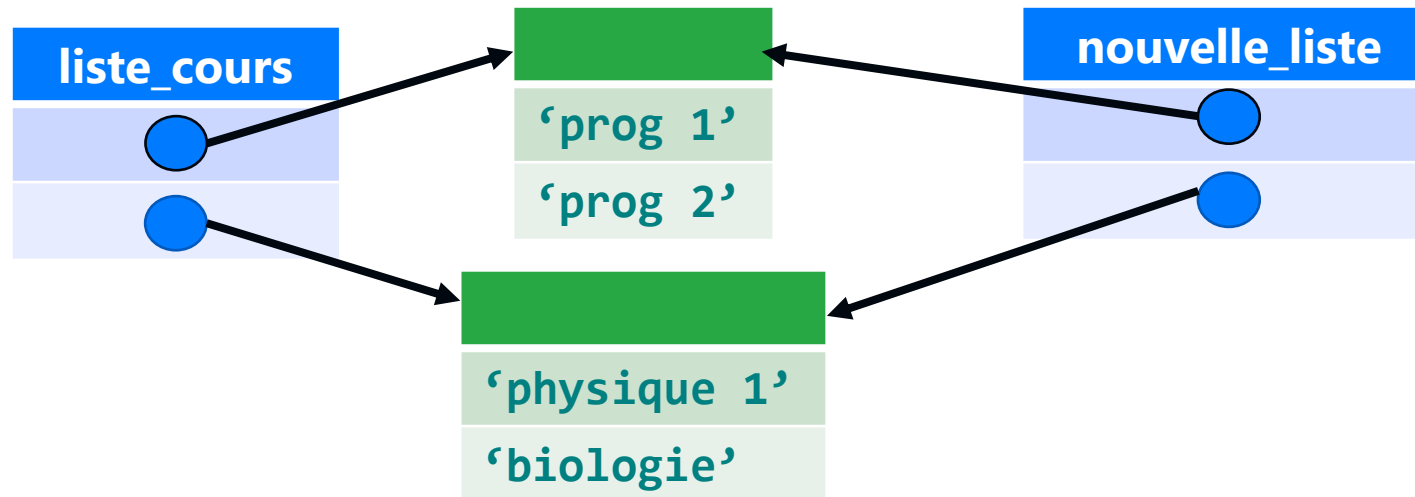
L'objet « `liste_cours` »
contient des pointeurs vers
deux autres objets listes.



Listes (méthode copy)

```
nouvelle_liste = liste_cours.copy()

print(nouvelle_liste)
[['prog 1', 'prog 2'], ['physique 1', 'biologie']]
```



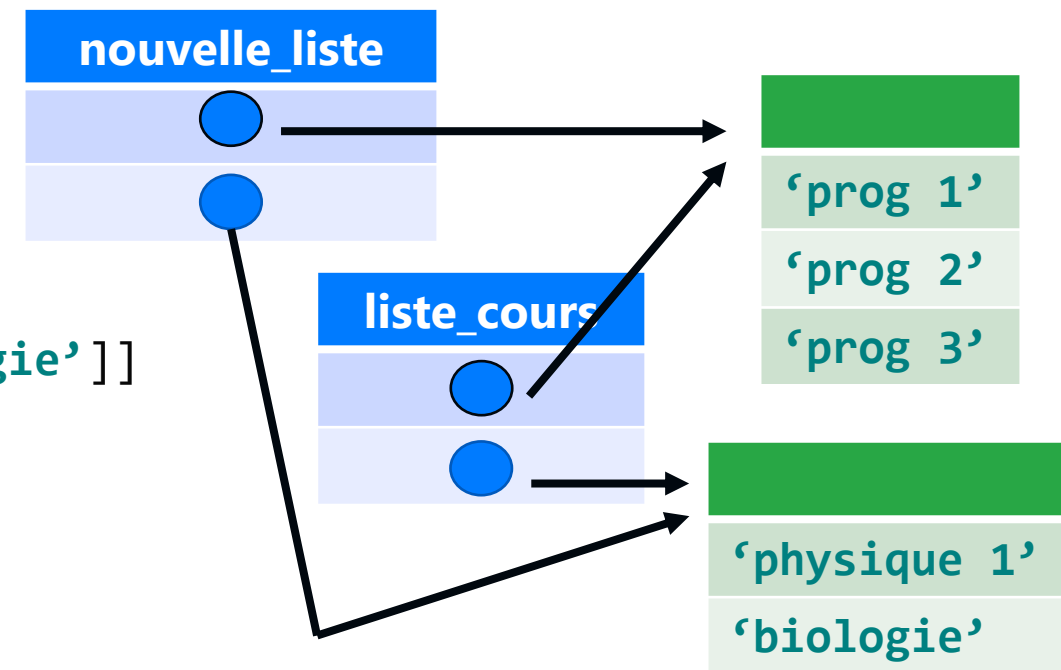
Listes (méthode copy)

- Les listes « `liste_cours` » et « `nouvelle_liste` » font référence aux mêmes objets. Donc des modifications effectuées sur un objet dans une des listes seront visibles à partir de l'autre liste.

```
liste_cours[0].append('prog 3')
```

```
print(nouvelle_liste)
```

```
[[ 'prog 1', 'prog 2', 'prog 3'], [ 'physique 1', 'biologie' ]]
```



Sous-Listes



- > Les listes peuvent être facilement divisées en sous-listes avec des opérations de « slicing »

```
cours = ['prog 1', 'prog 2', 'physique 1', 'biologie', 'anglais', 'philos']
```

- > `cours[x]` retourne la valeur à l'indice donné
- > `cours[x:y]` retourne une sous liste contenant les valeurs se trouvant de l'index [x] jusqu'à l'index [y-1]
- > `cours[-1]` retourne la valeur à la dernière position.

```
>>> print(cours[3])  
biologie
```

```
>>> print(cours[0:2])  
['prog 1', 'prog 2']  
...
```

```
>>> print(cours[-1])  
philos
```





Sous-Listes

```
cours = ['prog 1', 'prog 2', 'physique 1', 'biologie', 'anglais', 'philos']
```

> `cours[:x]` retourne une sous-liste contenant les valeurs du début jusqu'à l'index `[x]` (non inclus)

```
>>> print(cours[:2])  
['prog 1', 'prog 2']
```

> `cours[x:]` retourne une sous-liste contenant les valeurs de l'index `[x]` jusqu'à la fin

```
>>> print(cours[4:])  
['anglais', 'philos']
```

 > `cours[-x:-y]` retourne une sous-liste avec les valeurs de `-x` jusqu'à `-y` (non inclus)

```
>>> print(cours[-5:-2])  
['prog 2', 'physique 1', 'biologie']
```



Fonctions utiles avec les listes

- > Les fonctions « min() », « max() », et « sum() » sont souvent utilisées avec les listes

```
notes_gr10 = [ 34, 87, 96, 67.4, 72, 99.2, 59 ]
```

- > min() retourne la valeur la plus basse
- > max() retourne la valeur la plus élevée
- > sum() retourne la sommation des valeurs

```
>>> print(min(notes_gr10))  
5
```

```
>>> print(max(notes_gr10))  
99.2
```

```
>>> print(sum(notes_gr10))  
460.59999999999997
```

```
>>> moyenne = sum(notes_gr10) / len(notes_gr10)  
>>> print(moyenne)  
65.8
```

Boucles for et listes



retour



- Itère sur la liste, la variable `cours` prend la valeur de chacun des objets dans la liste un après l'autre.
- Itère sur la liste, la variable `index` prend les valeurs numériques de 0 jusqu'à la valeur de la `longueur` de la liste.

```
liste_cours = ['Programmation 1', 'Math',  
|...|...|...|...'Bureautique', 'Réseau 1', 'Math']
```

```
for cours in liste_cours:  
|...|print(cours)
```

```
Programmation 1  
Math  
Bureautique  
Réseau 1  
Math
```

```
for nbr, cours in enumerate(liste_cours):  
|...|print(f'Cours {nbr} : {cours}')
```

```
Cours 0 : Programmation 1  
Cours 1 : Math  
Cours 2 : Bureautique  
Cours 3 : Réseau 1  
Cours 4 : Math
```





Dictionnaire

Dictionnaires



- Similaire aux listes, les dictionnaires sont des structures de données qui peuvent stocker plusieurs valeurs.
- Les valeurs sont associées à une "clef" qui permet de récupérer les valeurs.

Clef : valeur

auto = { "marque": "Ford",
 "modele": "Mustang",
 "annee": 1964 }

Dictionnaires



Clef : valeur

```
auto = { "marque": "Ford",  
         "modele": "Mustang",  
         "annee": 1964 }
```

```
print(auto)  
# {'marque': 'Ford', 'modele': 'Mustang', 'annee': 1964}
```

```
print(auto['marque']) # utilisation de la clef pour obtenir sa valeur  
# Ford
```

```
print(f"{auto['marque']} {auto['modele']} {auto['annee']}")  
# Ford Mustang 1964
```

Dictionnaires - Méthodes



> `dict.get("clef")` retourne la valeur de la clef dans le dictionnaire

> `auto.get("modele")` → **"Mustang"**

Ne cause pas d'erreurs si la clef n'existe pas.

```
auto = { "marque": "Ford",  
         "modele": "Mustang",  
         "annee": 1964 }
```

> `dict["clef"] = valeur` Change la valeur correspondant à la clef. Si la clef n'existe pas, ajoute la paire clef:valeur

> `auto["annee"] = 1968`

> `auto["couleur"] = "rouge"`

```
auto → { "marque": "Ford",  
         "modele": "Mustang",  
         "annee": 1968 ,  
         "couleur": "rouge" }
```

Dictionnaires - Méthodes



- > `dict.pop("clef")` retire la paire clef:valeur du dictionnaire et retourne la valeur uniquement.
- > `annee_fabrication = auto.pop("annee")`

```
auto = { "marque": "Ford",  
         "modele": "Mustang",  
         "annee": 1964 }
```



```
auto == { "marque": "Ford",  
          "modele": "Mustang" }
```

```
annee_fabrication ➔ 1964
```



Débogage avec un IDE

Principe et outils dans Visual Studio Code

Messages d'erreur



- Important pour la maintenance de code complexe.
- Contiens plusieurs outils de débogage
- Problème avec le code :
 1. Regarder le message d'erreur dans la console

```
⊗ pierre-paul@pp-vm:~/scripts$ /bin/python3 /home/pierre-paul/scripts/zzz_factoriel_broken.py
Traceback (most recent call last):
  File "/home/pierre-paul/scripts/zzz_factoriel_broken.py", line 10, in <module>
    print (factoriel(a))
NameError: name 'a' is not defined
```

La ligne qui a
causé une erreur

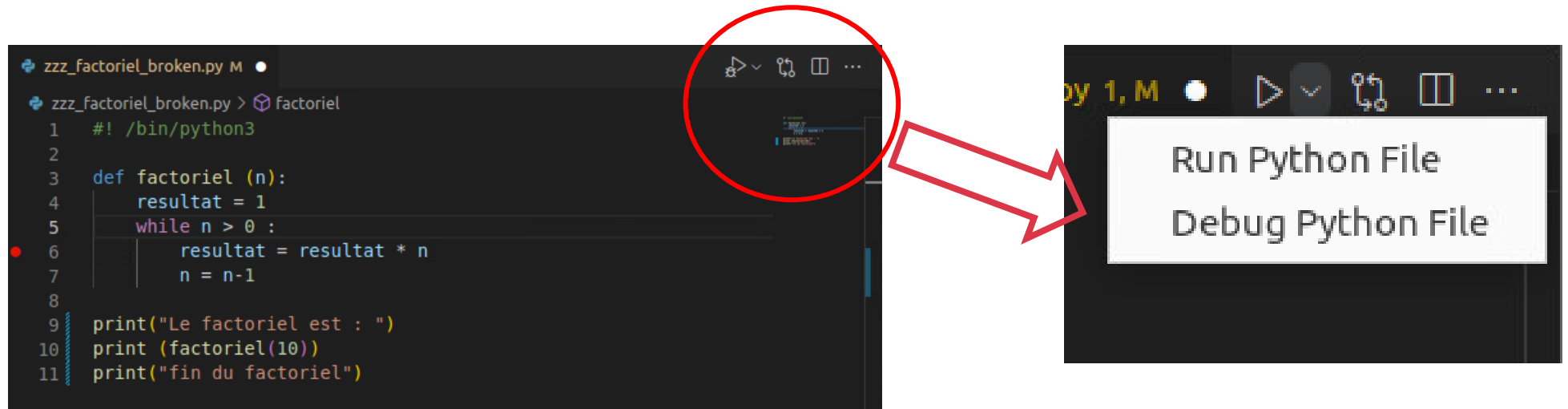
Le fichier qui a causé une erreur

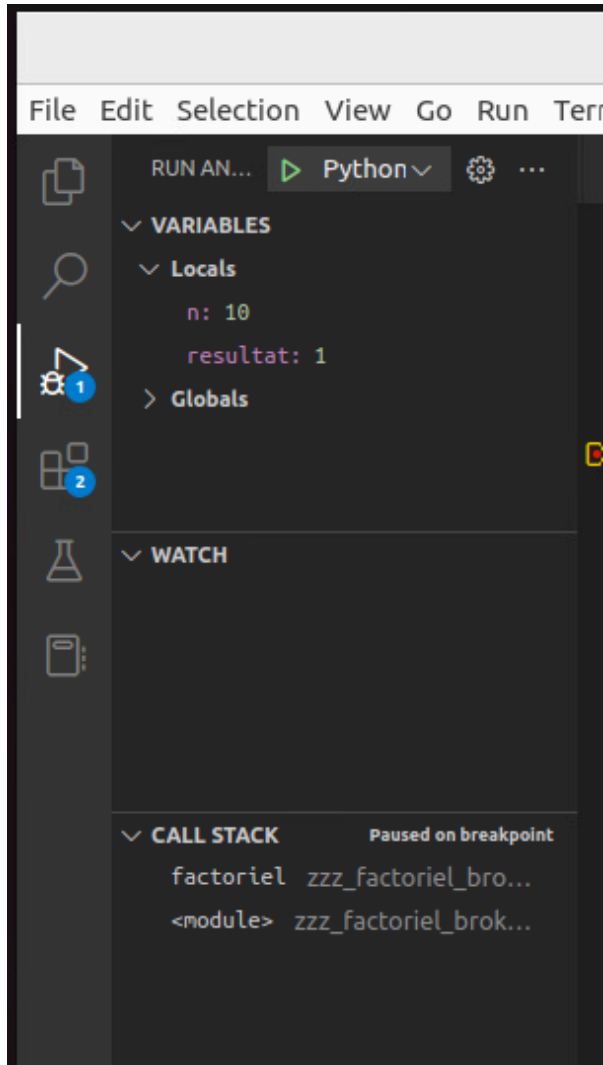
L'erreur qui a eu lieu. Ici la
variable « a » n'est pas définie.

> Problème avec le code :

2. Ajouter un ou des breakpoint(s) avec un clic dans l'espace à gauche des lignes de code

Quand on lance le script en mode Debug : le script va passer à travers les lignes de code et s'arrêter au premier breakpoint rencontré.





> Problème avec le code :







3. Une fois arrêté, on peut exécuter le code ligne par ligne avec le panneau de contrôle qui apparaît dans le mode débogage.

On peut voir les variables et leurs valeurs dans le panneau de débogage à gauche à mesure que le code est exécuté.



Naviguer le débogueur



-  ➤ Continue : Le code s'exécute jusqu'au prochain breakpoint
-  ➤ Step over : Exécute la prochaine ligne de code SANS entrer dans les fonctions
-  ➤ Step into : Exécute la prochaine ligne de code INCLUANT entrer dans les fonctions s'il s'agit de la prochaine ligne exécutée
-  ➤ Step out : Recule d'une ligne dans le code NE MODIFIE PAS LES VARIABLES
-  ➤ Restart : Relance le script (encore en mode débogage)
-  ➤ Stop : Quitte le mode débogage

Exécution de code spécifique

- Lorsqu'on développe un script, on peut décider d'exécuter uniquement certaines parties pour des fins de débogage.
- Dans le cas où une partie du code prend longtemps à exécuter ou n'est pas nécessaire

ATTENTION ! ON PASSE A LA CONSOLE PYTHON !

```
R02_Demo > /* R2_demo4 copy.py
1  #execution de lignes spécifiques
2
3  while True:
4      print("Boucle infinie")
5
6  cartes_graphiques_en_stock = ['GeForce RTX 3070Ti',
7                                'Radeon RX 6950 XT',
8                                'Radeon RX 6900 XT']
9  for carte in cartes_graphiques_en_stock:
10     print(f"La {carte} est disponible en magasin.")
11
```

PROBLÈMES 1 SORTIE CONSOLE DE DÉBOGAGE TERMINAL .NET INTERACT

Copyright (C) Microsoft Corporation. Tous droits réservés.

Testez le nouveau système multiplateforme PowerShell <https://aka.ms/powershell>

PS C:\Users\pierre-paul.gallant\Cégep Édouard-Montpetit\CMT-420_I
> & "C:/Program Files/Python310/python.exe"

Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC
Type "help", "copyright", "credits" or "license" for more informa
>>> cartes_graphiques_en_stock = ['GeForce RTX 3070Ti',
... 'Radeon RX 6950 XT',
... 'Radeon RX 6900 XT']
>>>
>>> for carte in cartes_graphiques_en_stock:
... print(f"La {carte} est disponible en magasin.")
...
La GeForce RTX 3070Ti est disponible en magasin.
La Radeon RX 6950 XT est disponible en magasin.
La Radeon RX 6900 XT est disponible en magasin.
>>>

Exécution de code spécifique



1. Sélectionner les lignes à exécuter.
2. Presser « F1 » pour voir toutes les commandes.
3. Trouver l'option « Run Selection/Line in python »

The screenshot shows a code editor with a Python script. The script is titled `R2_demo2.py` and contains the following code:

```
/* R2_demo2.py
/* R2_demo1.py
R02_Demo > /* R2_demo4 copy.py
1  #execution de lignes
2
3  while True:
4      print("Boucle infinie")
5  #
6  cartes_graphiques_en_stock = []
7  ...
8  ...
9  for carte in cartes_graphiques_en_stock:
10     print(f"La {carte} est disponible en magasin.")
11
12
13  while True:
14     print("Boucle infinie")
15
16  cartes_graphiques_rupture_de_stock = ['GeForce GTX 750Ti']
17  carte_en_rupture = cartes_graphiques_en_stock.pop()
18  cartes_graphiques_rupture_de_stock.append(carte_en_rupture)
19  print(f"La {carte_en_rupture} n'est pas disponible en magasin.")
20
21
```

A context menu is open over the code, showing the following options:

- Python: Exécuter la sélection/la ligne dans
- Python: Run Selection/Line in Python Terminal
- PowerShell: Run Selection
- Jupyter: Run Selection/Line in Interactive Notebook
- Python: Exécuter la sélection/la ligne dans
- Python: Run Selection/Line in Django Shell
- Terminal: Exécuter le texte sélectionné dans
- Terminal: Run Selected Text In Active Terminal

The menu is triggered by a right-click on the code. The first option, "Python: Exécuter la sélection/la ligne dans", is highlighted. The second option, "Python: Run Selection/Line in Python Terminal", is also visible. The third option, "PowerShell: Run Selection", is visible. The fourth option, "Jupyter: Run Selection/Line in Interactive Notebook", is visible. The fifth option, "Python: Exécuter la sélection/la ligne dans", is visible. The sixth option, "Python: Run Selection/Line in Django Shell", is visible. The seventh option, "Terminal: Exécuter le texte sélectionné dans", is visible. The eighth option, "Terminal: Run Selected Text In Active Terminal", is visible.

Exécution de code spécifique

- > « Run Selection/Line in python » lance l'interpréteur Python dans le terminal et lui passe les lignes sélectionnées
- > « shift » + « return » exécute la dernière commande de Visual Studio Code
- > On peut continuer d'exécuter les portions de code désirées avec l'interpréteur.

```
R02_Demo > /* R2_demo4 copy.py
1  #execution de lignes spécifiques
2
3  while True:
4      print("Boucle infinie")
5
6  cartes_graphiques_en_stock = ['GeForce RTX 3070Ti',
7                                'Radeon RX 6950 XT',
8                                'Radeon RX 6900 XT']
9  for carte in cartes_graphiques_en_stock:
10     print(f"La {carte} est disponible en magasin.")
11
```

PROBLÈMES 1 SORTIE CONSOLE DE DÉBOGAGE TERMINAL .NET INTERACT

Copyright (C) Microsoft Corporation. Tous droits réservés.

Testez le nouveau système multiplateforme PowerShell <https://aka.ms/powershell>

```
PS C:\Users\pierre-paul.gallant\Cégep Édouard-Montpetit\CMT-420_I
> & "C:/Program Files/Python310/python.exe"
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC
Type "help", "copyright", "credits" or "license" for more informa
>>> cartes_graphiques_en_stock = ['GeForce RTX 3070Ti',
...                                'Radeon RX 6950 XT',
...                                'Radeon RX 6900 XT']
>>>
>>> for carte in cartes_graphiques_en_stock:
...     print(f"La {carte} est disponible en magasin.")
...
La GeForce RTX 3070Ti est disponible en magasin.
La Radeon RX 6950 XT est disponible en magasin.
La Radeon RX 6900 XT est disponible en magasin.
>>>
```


Exemple

- Deux sections de code exécuté sans exécuter les boucles infinies

```
/* R2_demo2.py  /* R2_demo1.py  /* Ex4 List.py 1  /* R2_demo3.py  /* R2_demo4 copy.py
R02_Demo > /* R2_demo4 copy.py
1  #execution de lignes spécifiques
2
3  while True:
4      print("Boucle infinie")
5      ⚡
6  cartes_graphiques_en_stock = ['GeForce RTX 3070Ti',
7      'Radeon RX 6950 XT',
8      'Radeon RX 6900 XT']
9  for carte in cartes_graphiques_en_stock:
10     print(f"La {carte} est disponible en magasin.")
11
12
13  while True:
14     print("Boucle infinie")
15
16  cartes_graphiques_rupture_de_stock = ['GeForce GTX 750Ti']
17  carte_en_rupture = cartes_graphiques_en_stock.pop()
18  cartes_graphiques_rupture_de_stock.append(carte_en_rupture)
19  print(f"La {carte_en_rupture} n'est pas disponible en magasin pour l'instant.")
20
PROBLÈMES 1  SORTIE  CONSOLE DE DÉBOGAGE  TERMINAL  .NET INTERACTIVE  JUPYTER  AZURE
>>> cartes_graphiques_en_stock = ['GeForce RTX 3070Ti',
...                               'Radeon RX 6950 XT',
...                               'Radeon RX 6900 XT']
>>>
>>> for carte in cartes_graphiques_en_stock:
...     print(f"La {carte} est disponible en magasin.")
...
La GeForce RTX 3070Ti est disponible en magasin.
La Radeon RX 6950 XT est disponible en magasin.
La Radeon RX 6900 XT est disponible en magasin.
>>> cartes_graphiques_rupture_de_stock = ['GeForce GTX 750Ti']
>>> carte_en_rupture = cartes_graphiques_en_stock.pop()
>>> cartes_graphiques_rupture_de_stock.append(carte_en_rupture)
>>> print(f"La {carte_en_rupture} n'est pas disponible en magasin pour l'instant.")
La Radeon RX 6900 XT n'est pas disponible en magasin pour l'instant.
>>>
```



Zen of Python

- > **Beautiful** is better than **ugly**.
- > **Explicit** is better than **implicit**.
- > **Simple** is better than **complex**.
- > **Complex** is better than **complicated**.
- > **Flat** is better than **nested**.
- > **Sparse** is better than **dense**.
- > **Readability** counts.
- > **Special cases** aren't special enough to **break the rules**.
- > Although practicality beats **purity**.
- > **Errors** should never **pass silently**.
- > Unless **explicitly silenced**.
- > In the face of **ambiguity**, refuse the temptation to **guess**.
- > There should be **one** – and preferably only one – **obvious way** to do it.
- > Although that way may not be **obvious at first** unless you're Dutch.
- > **Now** is better than **never**.
- > Although **never** is often better than **right now**.
- > If the implementation is **hard to explain**, it's a **bad idea**.
- > If the implementation is **easy to explain**, it may be a **good idea**.
- > **Namespaces** are one honking **great idea** – let's do more of those!

