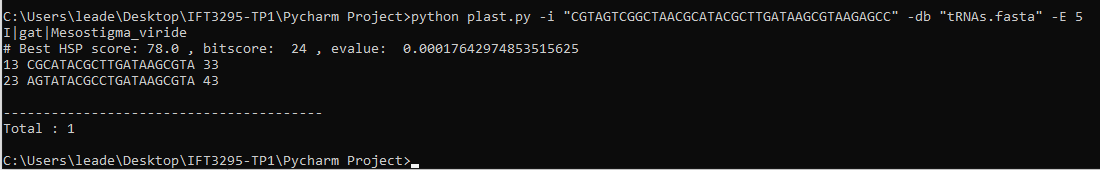
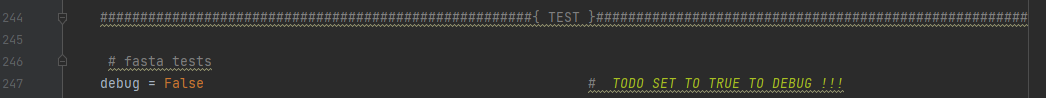
# 1. Résumé sur l’implémentation

Nous avons programmé PLAST dans le langage Python. Toute la logique se trouve dans le fichier **plast.py**. Cela dit, il est important que le fichier **FastaFile.py** soit dans le même projet puisque ce code gère l’ouverture des fichiers fasta. Enfin, l’installation de la librairie externe numpy est nécessaire pour le bon fonctionnement du programme.

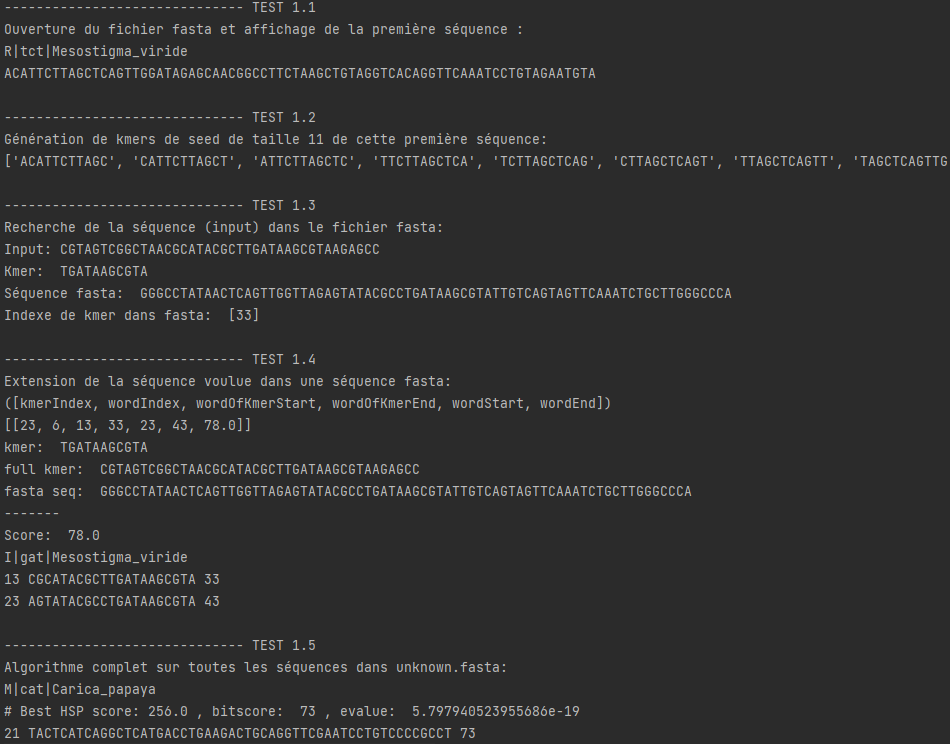
Nous avons utilisé l’IDE Pycharm pour écrire notre code. Vous pouvez l’utiliser aussi pour compiler le code ou lire nos commentaires. Sinon, il suffit d’exécuter une commande comme la suivante dans le dossier du projet :

python plast.py -i "CGTAGTCGGCTAACGCATACGCTTGATAAGCGTAAGAGCC" -db "tRNAs.fasta" -E 5

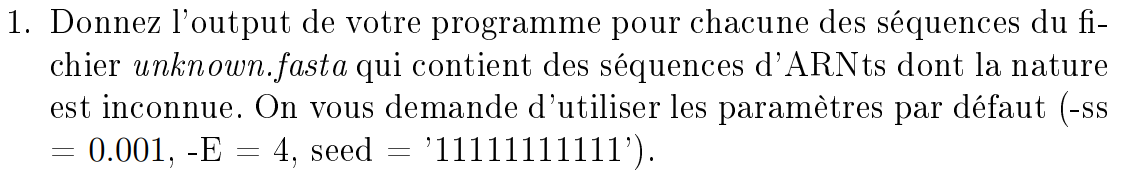
Le résultat à cette commande sur notre console est le suivant : 

Ce n’est pas demandé dans l’énoncé, mais il est aussi possible d’activer le mode debug de notre code en modifiant la ligne 247 de plast.py :   


Cela permet par exemple d’exécuter le programme plast sur chacune des séquences dans le fichier **unknown.fasta**. Biensûr, il n’est pas nécessaire de le faire puisque nous allons expliciter tous nos résultats dans la suite du rapport. Voici à quoi ressemble le debug :



# 2. Questions sur la mise en pratique



M|cat|Carica\_papaya

# Best HSP score: 256.0 , bitscore: 73 , evalue: 5.797940523955686e-19

21 TACTCATCAGGCTCATGACCTGAAGACTGCAGGTTCGAATCCTGTCCCCGCCT 73

21 GACTCATCAGGCTCATGACCTGAAGACTGCAGGTTCGAATCCTGTCCCCGCCT 73

R|tcg|Marchantia\_polymorpha

# Best HSP score: 297.0 , bitscore: 85 , evalue: 1.4155128232313686e-22

0 ACATCCTTAGCTCAGTAGGATAGAGCAACAGCCTTCTAAGCTGGTGGTCACAGGTTCAAATCCTGTAGGATG 71

0 GCATTCTTAGCTCA

GTTGGATAGAGCAACAACCTTCGAAGTTGATGGTCACAGGTTCAAATCCTGTAGGATG 71

P|tgg|Oryza\_sativa\_Japonica\_Group

# Best HSP score: 137.0 , bitscore: 40 , evalue: 4.980392986908555e-09

43 GAGGTCACGGGTTCAAATCCTGTCATCCCTA 73

43 AATGTCACGGGTTCAAATCCTGTCATCCCTA 73

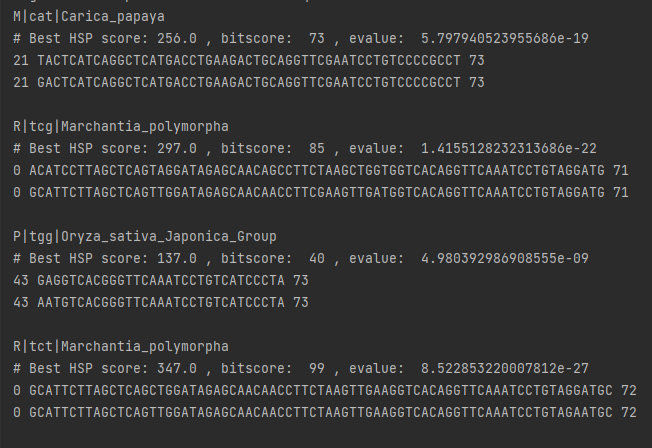
R|tct|Marchantia\_polymorpha

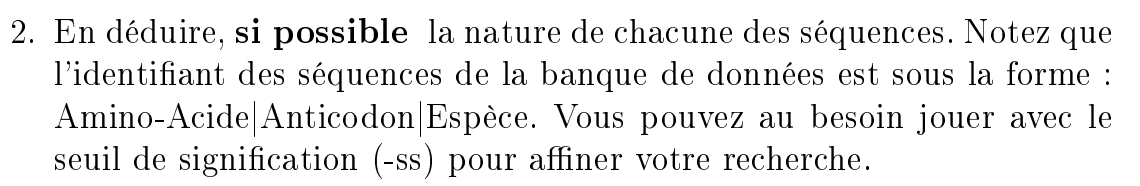
# Best HSP score: 347.0 , bitscore: 99 , evalue: 8.522853220007812e-27

0 GCATTCTTAGCTCAGCTGGATAGAGCAACAACCTTCTAAGTTGAAGGTCACAGGTTCAAATCCTGTAGGATGC 72

0 GCATTCTTAGCTCAGTTGGATAGAGCAACAACCTTCTAAGTTGAAGGTCACAGGTTCAAATCCTGTAGAATGC 72

Image de la console :



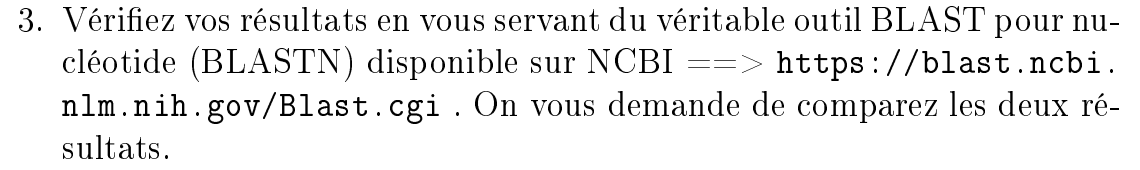


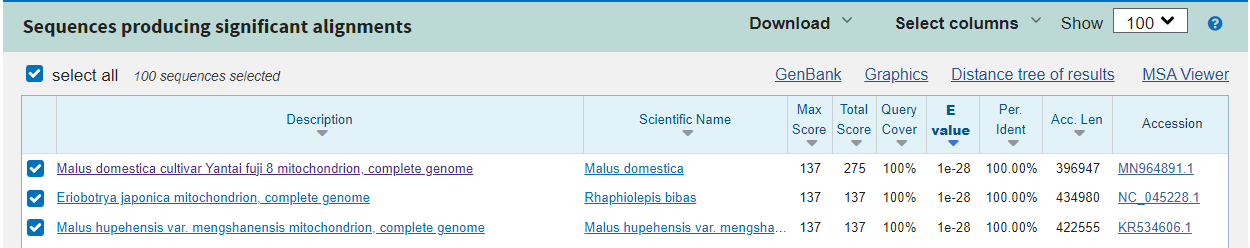
Première séquence : Amino-Acide : **M** | Anticodon : **cat**

Deuxième séquence : Amino-Acide : **R** | Anticodon : **tcg**

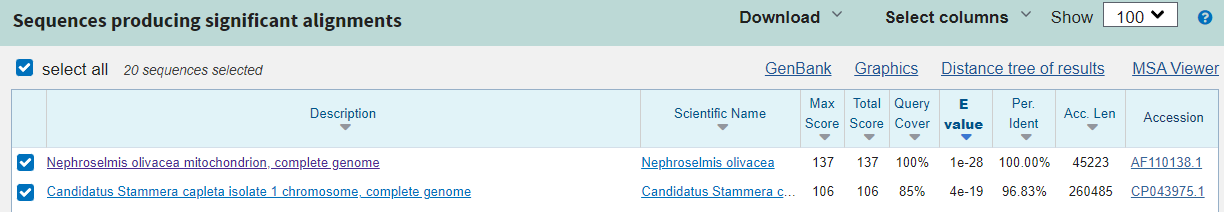
Troisième séquence : Amino-Acide : **P** | Anticodon : **tgg**

Dernière séquence  : Amino-Acide : **R** | Anticodon : **tct**

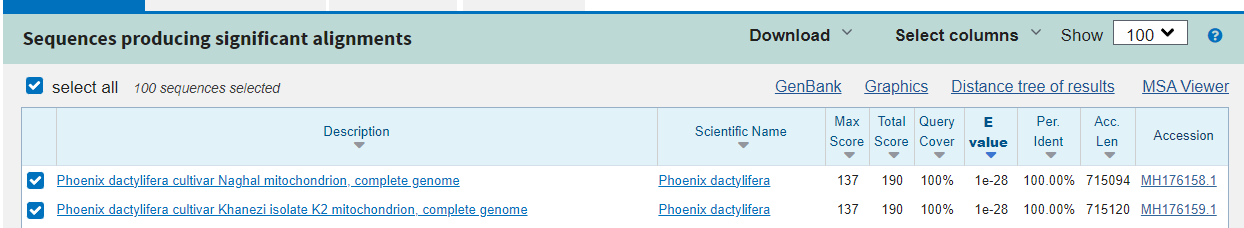


Première séquence : 

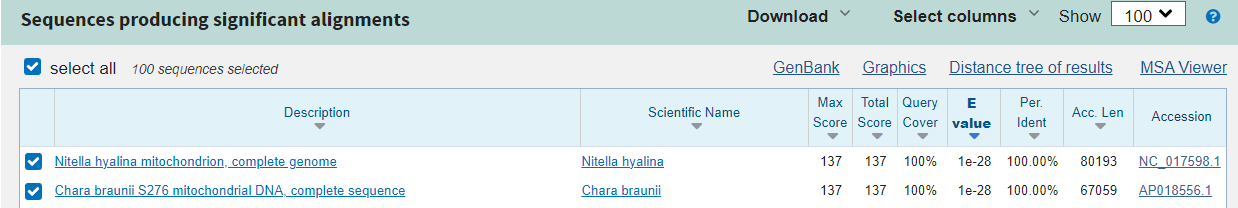
Nous avons obtenu un max score beaucoup trop grand comparé au score du véritable blast (256>137).

Deuxième séquence : 

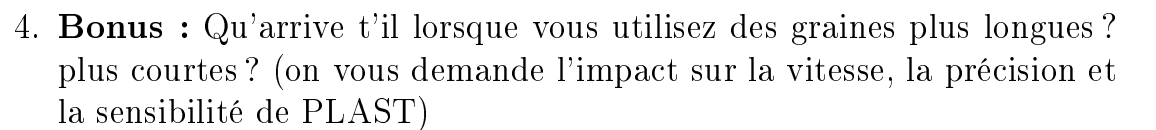
Encore une fois, le max score est beaucoup moins élevé que ce que nous avons obtenu (297>137)

Troisième séquence : 

Ici, nous avons obtenu le même max score (137). Cela dit, la E Value est différente.

Dernière séquence : 

Ici, notre score maximum est beaucoup trop élevé comparé au vrais score (347>137)



Suites à plusieurs tests sur notre programme, nous avons remarquer que l’exécution du programme est beaucoup plus lente avec des seeds plus court. En d’autres mots, réduire la taille des seed peut nuire aux performances (**vitesse**) du programme. Tandis qu’utiliser des seed plus grand améliore la performance du programme. Cela dit, il est raisonnable de croire qu’en utilisant des seeds trop grand, nous pouvons manquer certaines occurrences, mais ce n’est pas nécessairement négatif, puisque nous allons retenir seulement les sous-séquences les plus longues. Donc, cela peut améliorer la **précision** du programme.