# Titre

Programmation concurrente en Python

# Contenu validé

Oui

# Catch phrase (2 lignes max)

Cette formation a pour objectif de présenter diverses façons de paralléliser vos traitements en Python afin de tirer parti des architectures multi-cœurs ou multi-machines modernes.

# Description (10 lignes max)

Python est un langage puissant, orienté objet, convenant parfaitement à la majorité des développements informatiques.

Cette formation présente les diverses techniques de programmation concurrentes en Python comme le multithreading, le multiprocessing et la programmation asynchrone ainsi qu’une introduction aux traitements répartis sur plusieurs machines

Apprenez et mettez en œuvre des techniques de l’écosystème Python pour optimiser, accélérer et répartir vos traitements.

# Identifiant technique

pyconcurrent

# Domaine

développement

# Sous-domaine

Python

# Url

/formations/developpement/python-programmation-concurrente

# Ordre dans la page

5

# Prochaines sessions

## Lyon

21-22 juin 2021

## Lille

14-15 juin 2021

## Paris

28-29 juin 2021

# Équilibre théorie / pratique

35% théorie / 65% pratique

# Public visé / participants

* développeurs, chercheurs, ingénieurs confrontés à des problèmes d’optimisation de temps de traitement
* débutants s’abstenir

# Prérequis

* Bonne maîtrise et bonne pratique du langage Python
* Connaissance de la problématique de la parallélisation dans d’autres langages sera clairement un avantage

# Durée

2 jours, 14 heures

# Objectifs pédagogiques

* Découvrir et explorer les différentes techniques de traitements parallèles.
* Acquérir les concepts de la programmation parallèle, de synchronisation et de partages de données
* Sachez identifier les portions de programme qui sont parallélisables
* Sachez repérer les goulots d’étranglements dans votre code
* Identifier les parties de code consommatrices en temps de traitement, en ressources, qui sont candidates à des optimisations

# Équilibre théorie / pratique

# 25% théorie / 75% pratique

# Tarifs

1200 € HT par stagiaire (pauses et repas inclus)

# Plan de formation / programme

## Rappels et bibliothèques d’outils

### Mesurer les performances

* Analyser les temps de réponses
* Les profilers de la bibliothèque standard python (timeit, cprofile)
* Les profilers tiers
* Techniques de visualisation
* Examiner la consommation mémoire (gestion des structures complexes, le garbage collector, la bibliothèque externe memory\_profiler)

### Rappel sur la gestion des erreurs

* Concept et mise en œuvre des exceptions

TP : identification des goulots d’étranglement et optimisation des performances d’un programme simple

## Principe de programmation concurrente et première mise en œuvre

### Présentation des cas d’utilisation (découpage des temps de traitement, répartition de l’utilisation mémoire, gestion des taches longues ou bloquantes)

* Contrôler l’accès aux ressources
* Utiliser les temps d’attente liés aux entrées / sorties pour effectuer d’autres traitements
* Répartir la charge des traitements lourds, longs.

### Présentation des différents concepts et du vocabulaire

## Introduction à la programmation concurrente avec la bibliothèque « threading »

### Rappel sur la gestion mémoire

### Création de Threads en Python (fonctions et classes)

### La communication inter-tâche

* Principe
* Les outils : verrou, sémaphore, évènement, queue

### Cas d’usages et limitations

### Les traces en multi-tâche

### Arrêt propre des Threads

TP : création d’un ordonnanceur de threads, avec mise en place d’un arrêt propre des threads lancés

## La programmation multi-processus : la bibliothèque « multiprocessing »

### Cas d’usages et architecture map / reduce

### Notion de Pools

### Les traces en multi-processus

TP : Mise en œuvre d’un découpage de traitements longs puis d’agrégation des résultats

## La programmation asynchrone : la bibliothèque « asyncio »

### Principe et cas d’usage

### Les « futures », les tasks

### Le lancement des futures, la gestion des futures

### La boucle locale

### La gestion des tâches longues

TP : utilisation de techniques asynchrones pour télécharger des pages internets

## Initiation aux traitements répartis (sur plusieurs machines)

### Mise en œuvre en python natif :

* Avec la bibliothèque « multiprocessing » et les managers

### Autres mises en œuvre existantes

* La bibliothèque externe « RPyc»
* La bibliothèque externe « ray »
* Découverte des bibliothèques externes ~~“~~Dask”
* Allez plus loin (discussion autour de Spark, Airflow)

TP : mise en œuvre des différentes bibliothèques et restitution en groupes

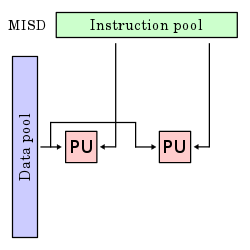
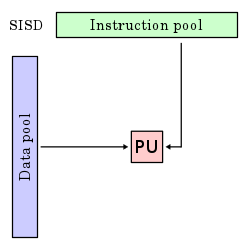
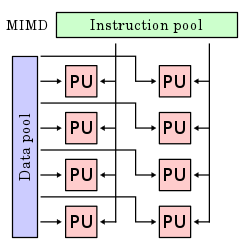
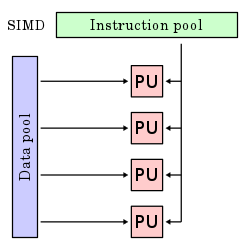
## Synthèse sur les cas d’usage

### Dans quels cas utiliser le multithreading, le multiprocessing, la programmation asynchrone et la programmation répartie ?

### Réflexions sur le SPMD (Single Program Multiple Data), MPSD (Multple Instructions Single Data)

### Discussion sur les temps incompressibles (loi d'Amdahl) et la séparabilité des données

# Liens et ressources

https://en.wikipedia.org/wiki/Flynn%27s\_taxonomy#Multiple\_programs,\_multiple\_data\_streams\_(MPMD)   
  
   
   
   


## 