FMS ACADEMY BY FMS

FACYLITIES MULTI SERVICES

Une entreprise ordinaire à l'agilité extraordinaire

FORMATION PYTHON AWS

PYTHON OBJET ET AVANCÉ

martial.bret@fms-ea.com

06 49 71 51 16

Version : 1.0

DMAJ : 2/10/23

FMS-EA 2023 © M. Bret – Tous droits réservés

Module : DEV-PYTH-002
Python

01 – Mettre en place et utiliser un environnement virtuel
02 – Découper un programme Python en paquets et modules
03 – Manipuler des itérateurs et des générateurs
04 – Utiliser la programmation orientée objet en Python
05 – Créer et utiliser un décorateur
06 – Vérifier statiquement les types d'un programme Python
07 – Utiliser des *threads*
08 – Réaliser une IHM *desktop* en Python

FMS-EA 2023 © M. Bret – Tous droits réservés

Sommaire

- Environnements virtuels
- Modularité
- Itérateurs, générateurs
- Programmation orientée objet
- Décorateurs
- Typage statique
- Threads
- Tkinter
- Ressources
- Pour aller plus loin

FMS-EA 2023 © M. Bret – Tous droits réservés

Environnements virtuels 1/2

- Isolation des paquets (et leur version) utilisés par chaque projet
- Création, à l'aide du module venv :

```
> python -m venv venv-projet
```

```
> python -m venv --system-site-packages venv-projet
```

- Activation de l'environnement virtuel :

```
> .\venv-projet\Scripts\activate
(venv-projet) > pip list
Package    Version
---------- -------
pip        23.2.1
setup Tools 65.5.0
```

- Désactivation de l'environnement virtuel :

```
(venv-projet) > deactivate
>
```

FMS-EA 2023 © M. Bret – Tous droits réservés

Environnements virtuels 2/2

- PyCharm :
  - Création (et utilisation) d'un environnement virtuel : *File > New Project…*
    - *Location:* répertoire du projet
    - *Python Interpreter: > New environment using* Virtualenv >
      - *Location:* ex. : dans le répertoire parent du répertoire du projet
      - *Base Interpreter:* Python 3.11
      - *Inherit global site-packages ?*
      - *Make available to all projects ?*
  - Utilisation d'un environnement existant :
    - clic interpréteur (en bas à droite) > *Add new interpreter > Add local interpreter >*
      - *Environment:* Existing
      - *Interpreter:* … > `venv-projet` > `Scripts` > `python.exe`
  - *File > Settings > Project: … > Python Interpreter*

FMS-EA 2023 © M. Bret – Tous droits réservés

Modularité

- *Teams*/1-cours/1-master_class/combinatoire.py

```
from factorielle import factorielle

def arrangements(k, n):
    """Fonction retournant le nombre de possibilités de choisir
       k objets parmi n, en tenant compte de l'ordre."""
    return factorielle(n) // factorielle(n - k)

def combinaisons(k, n):
    """Fonction retournant le nombre de possibilités de choisir
       k objets parmi n, sans tenir compte de l'ordre."""
    return factorielle(n) // (factorielle(k) * factorielle(n - k))

if __name__ == '__main__':
    print("Nombre de possibilités de choix d'une poule de " +
          "5 équipes tirées parmi un ensemble de 20 :")
    print(combinaisons(5, 20)) # 15504

    print("Nombre d'arrivées différentes d'une course de tiercé " +
          "comprenant 12 chevaux :")
    print(arrangements(3, 20)) # 6840
```

FMS-EA 2023 © M. Bret – Tous droits réservés

Itérateurs, générateurs

Itérateur

```
pioche = [('8', '♡'), ('9', '♠'), ('10', '♢'), ('V', '♣'), ('A', '♡')]
it_pioche = iter(pioche)
carte_j1 = next(it_pioche)  # ('8', '♡')
carte_j2 = next(it_pioche)  # ('9', '♠')
```

Générateur

```
def fibonacci():
    """Génère les nombres de la suite (infinie) de Fibonacci"""
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b
```

```
# affiche les nombres de la suite de Fibonacci < 1000
suite_fibonacci = fibonacci()
for n in suite_fibonacci:
    if n >= 1000:
        break
    print(n, " ", end="")
```

```
# affiche les nombres de la suite de Fibonacci < 1000
suite_fibonacci = fibonacci()
n = next(suite_fibonacci)
while n < 1000:
    print(n, " ", end="")
    n = next(suite_fibonacci)
```

Expression génératrice

```
fibo = (n for n in fibonacci() if n >= 1000)
next(fibo)  # 1597
```

FMS-EA 2023 © M. Bret – Tous droits réservés

Programmation orientée objet 1/2


```python
class Forme:
    formes = []

    @classmethod
    def nb_formes(cls):
        return len(cls.formes)

    def __init__(self, position):
        self.position = position
        Forme.formes.append(self)

    def __repr__(self):
        nom = type(self).__name__
        return f"{nom}{self.position}"
```

```python
class Cercle(Forme):

    def __init__(self, position, rayon):
        super().__init__(position)
        self.rayon = rayon

    def perimetre(self):
        return 2 * math.pi * self.rayon
```

```python
class Rectangle(Forme):
    def __init__(self, position, largeur, hauteur):
        super().__init__(position)
        self.largeur = largeur
        self.hauteur = hauteur

    def perimetre(self):
        return self.largeur * self.hauteur
```

```python
c1 = Cercle((0, 0), 20)
c2 = Cercle((10, 10), 10)
r1 = Rectangle((5, 5), 20, 10)
r2 = Rectangle((10, 5), 5, 10)
print(f"{Forme.nb_formes()} formes, leur périmètre :")
for forme in Forme.formes:
    perimetre = forme.perimetre()
    print(f"- {forme} : {perimetre}")
```

```
4 formes, leur périmètre :
- Cercle(0, 0) : 125.66370614359172
- Cercle(10, 10) : 62.83185307179586
- Rectangle(5, 5) : 200
- Rectangle(10, 5) : 50
```

2023 © M. Bret – Tous droits réservés

Décorateurs

- Décorateur mémorisant les résultats d'une fonction

```
def memorise(f):
    valeurs_f = {}

    def fonction_decoree(n):
        valeurs_f_n = valeurs_f.get(n)
        if valeurs_f_n is None:
            valeurs_f_n = valeurs_f[n] = f(n)
        return valeurs_f_n

    return fonction_decoree
```

```
@memorise
def factorielle(n):
    return n * factorielle(n - 1) if n > 1 else 1
```

```
@memorise
def fibonacci(n):
    if n < 2:
        return n
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```

- sans décorateur :
  - timeit(lambda: factorielle(80), number=100000) : ~ 0,5 sec.
  - timeit(lambda: fibonacci(40), number=1) : ~ 15 sec.
- avec décorateur :
  - timeit(lambda: factorielle(80), number=100000) : ~ 0,008 sec.
  - timeit(lambda: fibonacci(40), number=1) : ~ 1,5 µsec.

FMS-EA 2023 © M. Bret – Tous droits réservés

Typage statique

- Repose sur l'utilisation du module [mypy]
- *Teams*/1-cours/1-master_class/factorielle_type.py

```python
def factorielle(n: int) -> int:
    if n < 2:
        return 1
    else:
        return n * factorielle(n - 1)
```

```python
print("Voici différentes valeurs de la fonction factorielle :")

nb: int | float
for nb in [4, 6.0, 8]:
    fact_nb: int = factorielle(nb)    ligne 17
    print(f"La factorielle de {nb} vaut {fact_nb}.")

fact_4 = factorielle('4')    ligne 20
print(f"La factorielle de 4 vaut {fact_4}.")
```

```
Voici différentes valeurs de la fonction factorielle :
La factorielle de 4 vaut 24.
La factorielle de 6.0 vaut 720.0.
```

```
> mypy .\factorielle_type.py
factorielle_type.py:17: error: Argument 1 to "factorielle" has incompatible type "float"; expected "int"  [arg-type]
factorielle_type.py:20: error: Argument 1 to "factorielle" has incompatible type "str"; expected "int"  [arg-type]
Found 2 errors in 1 file (checked 1 source file)
```

```
...
    if n < 2:
       ^^^^^
TypeError: '<' not supported between instances of 'str' and 'int'
```

FMS-EA 2023 © M. Bret – Tous droits réservés

Programmation orientée objet 2/2


```python
from typing import ClassVar, List, Tuple
from dataclasses import dataclass
from abc import ABC, abstractmethod
```

```python
@dataclass
class Forme(ABC):
    formes: ClassVar[List['Forme']] = []
    position: Tuple[float, float]

    @classmethod
    def nb_formes(cls) -> int:
        return len(cls.formes)

    def __post_init__(self):
        Forme.formes.append(self)

    @abstractmethod
    def perimetre(self) -> float:
        ...
```

```python
@dataclass
class Cercle(Forme):
    rayon: float

    def perimetre(self) -> float:
        return 2 * math.pi * self.rayon
```

```python
@dataclass
class Rectangle(Forme):
    largeur: float
    hauteur: float

    def perimetre(self) -> float:
        return self.largeur * self.hauteur
```

```python
c1 = Cercle((0, 0), 20)
c2 = Cercle((10, 10), 10)
r1 = Rectangle((5, 5), 20, 10)
r2 = Rectangle((10, 5), 5, 10)
print(f"{Forme.nb_formes()} formes, leur périmètre :")
for forme in Forme.formes:
    perimetre = forme.perimetre()
    print(f"- {forme} : {perimetre}")
```

```
4 formes, leur périmètre :
- Cercle(position=(0, 0), rayon=20) : 125.66370614359172
- Cercle(position=(10, 10), rayon=10) : 62.83185307179586
- Rectangle(position=(5, 5), largeur=20, hauteur=10) : 200
- Rectangle(position=(10, 5), largeur=5, hauteur=10) : 50
```

Threads 1/2

- Test en parallèle par chaque navire du tir du joueur

```
def analyze_shot(ship, shot_coord, analyze_shot_results, no_ship):
    logging.info("Thread %d: début", no_ship)
    is_hit = ship_is_hit(ship, shot_coord)
    analyze_shot_results[no_ship] = is_hit
    if is_hit:
        logging.info("Thread %d: navire touché !", no_ship)
        print('Un navire a été touché par votre tir !')
        ship[shot_coord] = False
        if ship_is_sunk(ship):
            print('Le navire touché est coulé !!')
            # le navire est supprimé de la flotte
            ships_list.remove(ship)
    time.sleep(random.randint(1,5))
    logging.info("Thread %d: fin", no_ship)
```

FMS-EA 2023 © M. Bret – Tous droits réservés

Threads 2/2

```
format = "%(asctime)s: %(message)s"
logging.basicConfig(format=format, level=logging.INFO,
                    datefmt="%H:%M:%S")
played_shots = set()  # ensemble des coordonnées des tirs des
joueurs
while ships_list:
    display_grid()
    next_shot_coord = ask_coord()
    played_shots.add(next_shot_coord)
    analyze_shot_results = [None] * len(ships_list)
    analyze_shot_for_ship_list = []
    for no_ship, ship in enumerate(ships_list):
        analyze_shot_for_ship = \
            threading.Thread(target=analyze_shot,
                             args=(ship, next_shot_coord,
                                   analyze_shot_results, no_ship))
        analyze_shot_for_ship.start()
        analyze_shot_for_ship_list.append(analyze_shot_for_ship)
    logging.info("Attente de la fin des threads")
    for analyze_shot_for_ship in analyze_shot_for_ship_list:
        analyze_shot_for_ship.join()
    logging.info("Tous les threads sont terminés")
    if not any(analyze_shot_results):
        print("Votre tir est tombé dans l'eau")
    print()
```

FMS-EA 2023 © M. Bret – Tous droits réservés

Tkinter

- *A venir…*

FMS-EA 2023 © M. Bret – Tous droits réservés

Ressources

- https://docs.python.org/fr/3/tutorial
  - Environnements virtuels et paquets
    - https://code.visualstudio.com/docs/python/environments
  - Itérateurs, Générateurs, Classes
- https://frederic-lang.developpez.com/tutoriels/python/python-de-zero/
- https://python-course.eu/advanced-python/
  - List Comprehension, Generators and Iterators, Intro to Object Oriented Programming
- https://realpython.com/python-type-checking
- *data classes :*
  - https://docs.python.org/fr/3/library/dataclasses.html
  - https://realpython.com/python-data-classes
- https://realpython.com/python-ellipsis/
- https://realpython.com/intro-to-python-threading/

FMS-EA 2023 © M. Bret – Tous droits réservés

Pour aller plus loin

- https://fr.wikipedia.org/wiki/Table_de_hachage
- Tests
  - unittest – Framework de tests unitaires
    - (intégré de base à Python)
  - doctest – Tests interactifs en Python
    - (un autre module de test adoptant une approche très différente)
  - pytest – Framework de test
  - Cours "Testez votre projet Python«
    - (cours complet sur le sujet, incluant le test d'applications Web utilisant les framework Flask et Django)
- Programmation graphique avec Tkinter
  - Tutoriel : https://tkdocs.com/tutorial/index.html (sélectionner *Show: Python* pour ne voir que les exemples de code Python)
  - Documentation de référence : https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/index.html
  - Chapitre sur la documentation officielle de Python 3.10 : https://docs.python.org/fr/3/library/tk.html

FMS-EA 2023 © M. Bret – Tous droits réservés