كيف تستخدم التسجيل logging في بايثون 3

تأتي وحدة التسجيل (logging) مبدئيا توزيعة بايثون المعيارية وتقدم حلا لمتابعة الأحداث التي تحصُل أثناء عمل البرمجية. تستطيع إضافة استدعاءات التسجيل للشيفرة البرمجية الخاصة بك للإشارة لما تحقق من أحداث.

تسمح لك وحدة التسجيل بإعداد كل من التسجيلات التشخيصية التي تسجل الأحداث المقترنة بعمليات التطبيق، بالإضافة إلى تسجيلات التدقيق التي تسجل الأحداث الخاصة بحركات المستخدم بهدف تحليلها.

وحدة التسجيل هذه مختصة في حفظ السجلات في ملفات حافظ وحدة التسجيل على سجل الأحداث التي تحدث خلال عمل البرنامج، مما يجعل من رؤية مخرجات البرنامج المتعلقة بأحداثه أمراً متاحاً.

قد يكون استخدام الأمر `print` أمرا مألوفا لك خلال الشيفرة البرمجية لفحص الأحداث. يقدم الأمر `print` طريقة بدائية لإجراء عملية التنقيح الخاصة بحل المشاكل خلال عمل البرمجية. بينما يعد تضمين تعليمات `print` خلال الشيفرة البرمجية طريقة لمعرفة مسار تنفيذ البرنامج والحالة الحالية له، إلا أن هذه الطريقة أُثبت أنها أقل قدرة على الصيانة من استخدام وحدة التسجيل في بايثون، وذلك للأسباب التالية:

- باستخدام تعليمات `print` يصبح من الصعب التفرقة بين مخرجات البرنامج الطبيعية وبين مخرجات التنقيح لتشابههما.
- عندما تنتشر تعليمات `print` خلال الشيفرة البرمجية، فإنه لا توجد طريقة سهلة لتعطيل التعليمات الخاصة بالتنقيح.
- من الصعب إزالة كافة تعليمات `print` عندما تنتهي من عملية التنقيح.
- لا توجد سجلات تشخيصية للأحداث.

من الجيد البدء بالتعود على استخدام وحدة التسجيل المعيارية في بايثون خلال كتابة الشيفرة البرمجية بما أنها طريقة تتلاءم أكثر مع التطبيقات التي يكبر حجمها عن سكريبتات بايثون بسيطة، وكذلك بما أنها تقدم طريقة أفضل للتنقيح.

إذا كنت متعودا على استخدام تعليمات `print` لرؤية ما يحدث في برنامجك خلال العمل، فمن المحتمل مثلا أنك تعودت على رؤية برنامج يُعرف صنفًا Class وينشئ منه عناصر كما في المثال التالي:

```
class Pizza():
    def __init__(self, name, price):
        self.name = name
        self.price = price
```

```
        print("Pizza created: {}
(${})".format(self.name, self.price))

    def make(self, quantity=1):
        print("Made {} {} pizza(s)".format(quantity,
self.name))

    def eat(self, quantity=1):
        print("Ate {} pizza(s)".format(quantity,
self.name))

pizza_01 = Pizza("artichoke", 15)
pizza_01.make()
pizza_01.eat()

pizza_02 = Pizza("margherita", 12)
pizza_02.make(2)
pizza_02.eat()
```

توجد في الشيفرة السابقة الدالة `__init__` التي تستخدم لتعريف خصائص `name` و `price` للصنف `Pizza` كما تحتوي على الدالتين `make` لصنع البيتزا، و `eat` لأكلها وتأخذان المعطى `quantity` ذا القيمة الافتراضية 1.