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.

.

لنشغل البرنامج:

```
>> python pizza.py
```

وسنحصل على المخرج التالي:

```
Output
Pizza created: artichoke ($15)
Made 1 artichoke pizza(s)
Ate 1 pizza(s)
Pizza created: margherita ($12)
Made 2 margherita pizza(s)
Ate 1 pizza(s)
```

تسمح لنا تعليمات `print` برؤية أن البرنامج يعمل، ولكننا نستطيع أن نستخدم وحدة التسجيل لذات الغرض بدلا من ذلك.
لنقم بإزالة تعليمات `print` من الشيفرة البرمجية، ونستورد الوحدة باستخدام الأمر `import logging`:

```
import logging


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

المستوى التلقائي للتسجيل في وحدة التسجيل هو مستوى التحذير (WARNING) ، وهو مستوى فوق مستوى التنقيح (DEBUG). بما أننا سنستخدم الوحدة بغرض التنقيح في هذا المثال، سنحتاج الى تعديل إعدادات

التسجيل لتصبح بمستوى التنقيح logging.DEBUG بحيث تعود معلومات التنقيح لنا من خلال لوحة التحكم. ونقوم بإعداد ذلك بإضافة ما يلي بعد تعليماتة الاستيراد:

```
import logging

logging.basicConfig(level=logging.DEBUG)

class Pizza():
...
```

هذا المستوى المتمثل ب logging.DEBUG يشير لقيد رقمي قيمته 10.

سنستبدل الآن جميع تعليمات print

بتعليمات logging.debug() ، logging.DEBUG ثابت)

بينما logging.debug() دالة). نستطيع أن نمرر لهذه الدالة نفس المدخلات النصية لتعليماتة print كما هو موجود بالأسفل:

```
import logging

logging.basicConfig(level=logging.DEBUG)


class Pizza():
    def __init__(self, name, price):
        self.name = name
        self.price = price
        logging.debug("Pizza created: {}
(${})".format(self.name, self.price))

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

    def eat(self, quantity=1):
```

```
        logging.debug("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()
```

وسنحصل على المخرج `python pizza.py` لهذا الحد، نستطيع تشغيل البرنامج عبر تنفيذ الأمر التالي

```
Output
DEBUG:root:Pizza created: artichoke ($15)
DEBUG:root:Made 1 artichoke pizza(s)
DEBUG:root:Ate 1 pizza(s)
DEBUG:root:Pizza created: margherita ($12)
DEBUG:root:Made 2 margherita pizza(s)
DEBUG:root:Ate 1 pizza(s)
```

والتي تشير لمستوى `root` بالإضافة لكلمة `DEBUG` لاحظ أن مستوى التسجيل في المخرج السابق هو من الممكن أن يتم logging الذي يتم استخدامه. يعني ما سبق أن وحدة التسجيل (logger) المُسجل استخدامها لإعداد أكثر من مُسجل بأسماء مختلفة.

فمثلا، نستطيع إنشاء مسجلين باسمين مختلفين ومخرجات مختلفة كما هو موضح بالأسفل

```
logger1 = logging.getLogger("module_1")
logger2 = logging.getLogger("module_2")

logger1.debug("Module 1 debugger")
logger2.debug("Module 2 debugger")

Output
DEBUG:module_1:Module 1 debugger
DEBUG:module_2:Module 2 debugger
```

بعد أن أصبحت لدينا المعرفة اللازمة لكيفية استخدام الوحدة logging لطباعة الرسائل على وحدة التحكم، دعونا نكمل شرح الوحدة ونتعرف على كيفية استخدام الوحدة في طباعة الرسائل إلى ملف خارجي.

التسجيل في ملف

الغرض الأساسي للتسجيل هو حفظ البيانات في ملف وليس إظهار معلومات التسجيل على وحدة التحكم. يتيح لك التسجيل في ملف حفظ بيانات التسجيل مع مرور الوقت واستخدامها في عملية التحليل والمتابعة ولتحديد

ما تحتاجه من تغيير على الشيفرة البرمجية.
لجعل عملية التسجيل تحفظ التسجيلات في ملف، علينا أن
نعدّل `logging.basicConfig()` بحيث تحتوي على معطى لاسم الملف(`filename`) ،
وليكن مثلا: `test.log`

```
import logging

logging.basicConfig(filename="test.log",
level=logging.DEBUG)


class Pizza():
    def __init__(self, name, price):
        self.name = name
        self.price = price
        logging.debug("Pizza created: {}
(${})".format(self.name, self.price))

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

    def eat(self, quantity=1):
        logging.debug("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()
```

الشيفرة البرمجية هنا هي نفسها الموجودة سابقا عدا أننا أضفنا اسم الملف الذي سنقوم بحفظ التسجيلات فيه.
بمجرد تشغيلنا للشيفرة السابقة، سنجد في نفس المسار الملف `test.log`. لنفتحه باستخدام محرر النصوص) nano أو أي محرر نصوص من اختيارك:(

```
$ nano test.log
```

وسيكون محتويات الملف كالتالي:

```
DEBUG:root:Pizza created: artichoke ($15)
DEBUG:root:Made 1 artichoke pizza(s)
DEBUG:root:Ate 1 pizza(s)
DEBUG:root:Pizza created: margherita ($12)
DEBUG:root:Made 2 margherita pizza(s)
DEBUG:root:Ate 1 pizza(s)
```

المخرج السابق هو نفسه الذي حصلنا عليه في القسم السابق من المقال، غير أنه الآن في ملف باسم `test.log` وليس على الطرفية.
لنغلق المحرر، ونجر بعض التعديلات التالية على المتغيرين `pizza_01` و`pizza_02`:

```
...
# Modify the parameters of the pizza_01 object
pizza_01 = Pizza("Sicilian", 18)
pizza_01.make(5)
pizza_01.eat(4)

# Modify the parameters of the pizza_02 object
pizza_02 = Pizza("quattro formaggi", 16)
pizza_02.make(2)
pizza_02.eat(2)
```

عند تنفيذ الشيفرة بعد حفظ التعديلات، ستُضاف التسجيلات الجديدة للملف وسيكون محتواه كالتالي:

```
DEBUG:root:Pizza created: artichoke ($15)
DEBUG:root:Made 1 artichoke pizza(s)
```

```
DEBUG:root:Ate 1 pizza(s)
DEBUG:root:Pizza created: margherita ($12)
DEBUG:root:Made 2 margherita pizza(s)
DEBUG:root:Ate 1 pizza(s)
DEBUG:root:Pizza created: Sicilian ($18)
DEBUG:root:Made 5 Sicilian pizza(s)
DEBUG:root:Ate 4 pizza(s)
DEBUG:root:Pizza created: quattro formaggi ($16)
DEBUG:root:Made 2 quattro formaggi pizza(s)
DEBUG:root:Ate 2 pizza(s)
```

تُعد البيانات الموجودة في الملف مفيدة، ولكننا نستطيع جعلها أكثر إعلاماً بإضافة بعض الإعدادات. بشكل أساسي، فإننا نريد أن نجعل السجلات مفصلة أكثر بإضافة الوقت الذي أنشئ السجل فيه.

نستطيع إضافة المعطى المسمى `format` ونضيف له النص `%(asctime)s` الذي يشير للوقت، كذلك، للإبقاء على ظهور مستوى التسجيل في السجلات، لابد أن نضيف النص `%(levelname)s` بالإضافة للسجل نفسه `%(message)s`.

لابد من الفصل بين كل خيار في المعطى `format` بالعلامة `:` كما هو موضح بالأسفل:

```
import logging

logging.basicConfig(
    filename="test.log",
    level=logging.DEBUG,
    format="%(asctime)s:%(levelname)s:%(message)s"
    )
.......
```

عندما ننفذ الشيفرة السابقة، سنحصل على تسجيلات جديدة في ملف `test.log` تتضمن الوقت الذي أنشئ فيه التسجيل بالإضافة لمستوى التسجيل ورسالة التسجيل:

```
Output
DEBUG:root:Pizza created: Sicilian ($18)
DEBUG:root:Made 5 Sicilian pizza(s)
DEBUG:root:Ate 4 pizza(s)
```

```
DEBUG:root:Pizza created: quattro formaggi ($16)
DEBUG:root:Made 2 quattro formaggi pizza(s)
DEBUG:root:Ate 2 pizza(s)
2017-05-01 16:28:54,593:DEBUG:Pizza created: Sicilian
($18)
2017-05-01 16:28:54,593:DEBUG:Made 5 Sicilian pizza(s)
2017-05-01 16:28:54,593:DEBUG:Ate 4 pizza(s)
2017-05-01 16:28:54,593:DEBUG:Pizza created: quattro
formaggi ($16)
2017-05-01 16:28:54,593:DEBUG:Made 2 quattro formaggi
pizza(s)
2017-05-01 16:28:54,593:DEBUG:Ate 2 pizza(s)
```

تبعا لاحتياجاتك، من الممكن أن تضيف إعدادات أخرى للمسجل بحيث تجعل التسجيلات التي يتم حفظها في الملف مرتبطة بك نوعا ما.

التنقيح بواسطة التسجيل في ملفات خارجية يتيح لك فهما شاملا لبرنامج بايثون مع مرور الوقت، معطيا الفرصة لحل المشاكل التي تظهر وتغيير ما تحتاج تغييره في الشيفرة البرمجية استنادا لما لديك من بيانات تسجيلات تاريخية وأحداث وحركات تمت خلال عمل البرنامج.