**Chương 10  
Lặp lại: Vòng lặp để xử lý địa lý**

**LỜI MỞ ĐẦU**

Những tiến bộ trong công nghệ cho phép thu thập và lưu trữ một lượng  
lớn dữ liệu GIS. Để học hỏi từ dữ liệu này, chúng ta cần tiến hành phân tích một cách hiệu quả. Xử lý hàng loạt là một khả năng viết kịch bản mạnh mẽ, tiết kiệm thời gian bằng cách tự động hóa các tác vụ lặp đi lặp lại. Chương 8 đã thảo luận về ba cấu trúc quy trình làm việc cơ bản (trình tự, ra quyết định và lặp lại) về mã giả và Chương 9 đã trình bày cú pháp Python để ra quyết định và mô tả dữ liệu GIS. Chương này tập trung vào cấu trúc lặp lại Python và cú pháp lặp, Python **For**-loops và **While**-loops, lặp với hàm phạm vi để xử lý địa lý, lặp lồng nhau và liệt kê nội dung thư mục. Sau đó chương kết thúc với một mẹo về cách gỡ lỗi các trục trặc khoảng trắng.

**Mục tiêu chương**

Sau khi đọc chương này, bạn sẽ có thể làm như sau:  
• Thực hiện **For**-loops và **While**-loops trong Python.  
• Xác định ba thành phần biến lặp chính trong một **While**-loop.  
• Giải thích cách Python **For**-loops hoạt động.  
• Sửa chữa các vòng lặp vô hạn.  
• Gọi một công cụ xử lý địa lý trong một **While**-loop để thay đổi một số tham số.  
• Tự động tạo danh sách số.  
• Vòng lặp với **range** hàm số.  
• Nhánh và vòng lặp trong các vòng lặp.  
• Liệt kê các tệp trong một thư mục.  
• Xử lý địa lý từng tệp trong danh sách.  
• Sửa lỗi thụt đầu dòng.

**10.1 Cú pháp vòng lặp**

Python có hai cấu trúc để thực hiện lặp lại, **For**-loops và **While**-loops, sử dụng các từ khóa Python **while** và **for**. Họ cung cấp hai cách tiếp cận hơi khác nhau. **For**-loops được sử dụng thường xuyên hơn trong các tập lệnh xử lý địa lý; tuy nhiên, cả hai kỹ thuật lặp đều được sử dụng, vì vậy sẽ rất hữu ích nếu bạn làm quen với cả hai.

với **While**-loops cung cấp giới thiệu dễ dàng hơn về vòng lặp vì Python, **While**-loops có các cơ chế lặp tiếp xúc; trong khi đó, hoạt động của **For**-loops ít rõ ràng hơn. Nhiều ngôn ngữ lập trình có chung **While**-loop với Python; trong khi, cấu trúc **For**-loops trong Python ít phổ biến hơn; cả hai sẽ có vẻ dễ dàng, một khi bạn làm quen được cú pháp của nó.

***10.1.1*** *WHILE****-Loops***  
Code trong ví dụ 10.1 sử dụng Python **WHILE**-loop và dòng được hiển thị ở bên trái. Bạn có thể dự đoán những gì kịch bản này sẽ in?

**Ví dụ 10.1**

|  |  |
| --- | --- |
| 123456 | *# simpleWhileLoop.py* x = 1 **while** x <= 5:  **print** (x)  x = x + 1 **print** (‘I’m done!’) |

MỘT WHILE-loop kiểm tra một biểu thức Boolean được đặt trên cùng một dòng với **while** từ khóa. Trong ví dụ 10.1, nó kiểm tra nếu x <= 5. 'Điều kiện', giống như những điều kiện được sử dụng trong các câu lệnh ra quyết định, đánh giá là đúng hoặc sai và các câu lệnh trong khối mã thụt lề tiếp theo (chẳng hạn như dòng 4 và 5) chỉ được thực hiện miễn là điều kiện là thật. Sau khi chạy bước thụt lề cuối cùng, nó sẽ đánh giá lại giá trị của điều kiện (true or false?). Nếu điều kiện là đúng, nó lặp lại các bước thụt lề từ trên xuống một lần nữa (dòng 4, sau đó đến dòng 5). Nếu điều kiện sai, nó sẽ di chuyển đến dòng mã đầu tiên **WHILE-loop** (dòng 6). Chúng tôi gọi các dòng mã thụt lề là ‘**WHILE**-loop code block’. WHILE-loop lặp đi lặp lại chạy khối mã này miễn là điều kiện kiểm tra vẫn đúng. Nếu điều kiện không bao giờ đúng ngay từ đầu, khối mã thụt lề sẽ không bao giờ được thực thi. Chạy 'simpleWhileLoop.py' và kết quả sẽ như thế này:

>>> 1

2

3

4

5

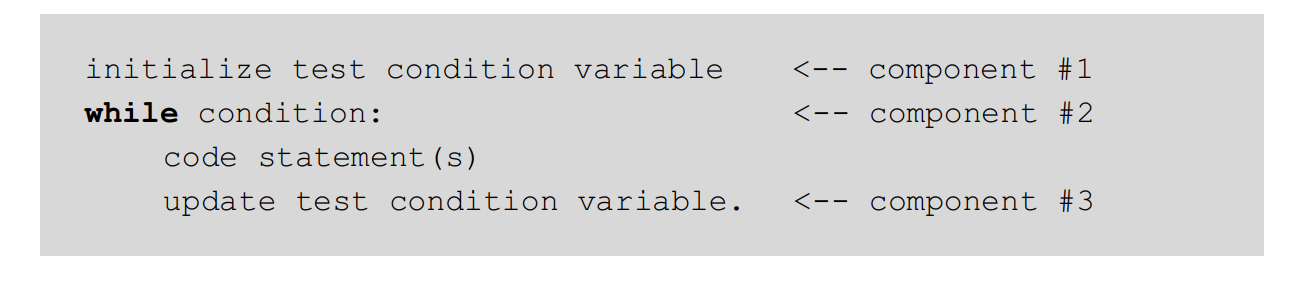
I’m done!

x được in năm lần và " I'm done! “ nằm ngoài WHILE-loop xong!" chỉ được in một lần, kể từ lần in này.

WHILE-loops dựa trên một điều kiện thử nghiệm. Nếu điều kiện đó không bao giờ trở thành sai, mã này có thể chạy trong một vòng lặp vô hạn. Trình thông dịch Python phải bị gián đoạn theo một cách đặc biệt nếu điều này xảy ra. Nhận xét dòng 5 trong Ví dụ 10.1 bằng cách đặt một dấu thăng (#) ở đầu dòng. Dự đoán những gì sẽ được in trước khi bạn chạy nó. Sau đó, hãy làm theo hướng dẫn 'Cách ngắt đoạn mã đang chạy'.

|  |
| --- |
| **Làm thế nào để làm gián đoạn mã chạy** 1. Đối với PythonWin,  (a) Nhấp chuột phải vào biểu tượng PythonWin đại diện cho quá trình đang chạy của bạn, ở góc dưới bên phải của màn hình.  (b) Nhấp vào “Break into running code”. Bạn có thể cần phải chọn điều này nhiều lần trước khi mã thực sự ngừng chạy.  (c) Sau khi bạn dừng mã đang chạy thành công, bản ghi lại 'KeyboardInterrupt' sẽ được in trong Cửa sổ Tương tác. 2. Trong PyScripter,  (a) Nhấp CTRL + F2  (b) Nhấp vào 'Có' trên hộp có nội dung 'Thông dịch viên đang bận. Bạn có chắc chắn muốn chấm dứt? ' |

Xóa dòng 5 khỏi Ví dụ 10.1 tạo ra 'vòng lặp vô hạn' vì điều kiện kiểm tra đang kiểm tra giá trị của x. Khi giá trị của x không được sửa đổi, điều kiện thử nghiệm không bao giờ trở thành sai. Nếu như x luôn luôn là 1, x <= 5 luôn luôn đúng! WHILE-loop kiểm tra vòng lặp thường liên quan đến một biến cần được khởi tạo bên ngoài vòng lặp và cập nhật bên trong vòng lặp. Thông thường, bản cập nhật sẽ xảy ra sau tất cả các lần WHILE-loop khối lặp. WHILE-loop thường có định dạng sau:

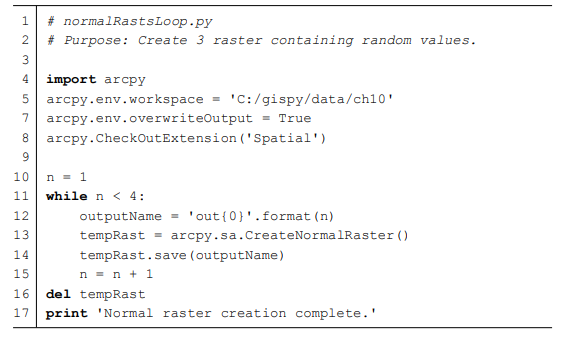


Nếu thiếu bất kỳ thành phần nào trong ba thành phần bao gồm (khởi tạo, kiểm tra và cập nhật) thì vòng lặp sẽ không hoạt động như mong đợi.

Chú ý các yếu tố cú pháp, dấu hai chấm và thụt đầu dòng. Tất cả các cấu trúc khối mã Python có chứa nhiều dòng mã liên quan — yêu cầu dấu hai chấm và thụt đầu dòng. Điều này bao gồm các khối mã có điều kiện, **WHILE** **FOR**-loop khối mã vòng lặp và các cấu trúc khác như hàm và lớp, sẽ được thảo luận trong các chương sắp tới. Dấu hai chấm luôn chỉ ra rằng một khối mã liên quan sẽ theo sau. Thiếu dấu hai chấm sẽ gây ra lỗi cú pháp. Lỗi cú pháp cũng sẽ xảy ra nếu khối mã trống hoặc nếu dòng đầu tiên sau dấu hai chấm không được thụt lề. Miễn là bạn gõ dấu hai chấm, việc thụt lề rất dễ dàng, vì IDE tự động thụt lề dòng tiếp theo khi bạn nhấn phím 'Enter' để chuyển sang dòng tiếp theo.

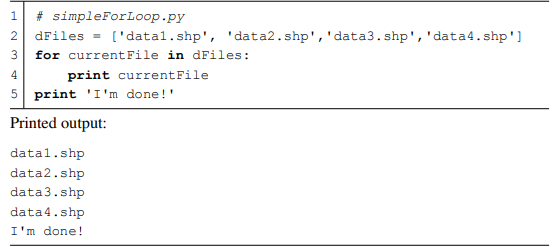
Ví dụ 10,2 Cho thấy một **WHILE**-loop ví dụ về xử lý địa lý vòng lặp với các số hàng ở bên trái. Tập lệnh này tạo ra ba raster chứa các giá trị ô ngẫu nhiên. Biến lặp,n, được khởi tạo, kiểm tra và cập nhật trên các dòng 10, 11 và 15. n được sử dụng để tạo một tên duy nhất cho mỗi raster đầu ra. Tên dữ liệu đầu ra được đặt ở đầu khối mã vòng lặp trên dòng 12. Biến lặp, n, bắt đầu với giá trị 1, sau đó trở thành 2 và cuối cùng là 3. Dòng 12 tạo tên **out1**, **out2**, và **out3**. Lưu ý rằng các bước thiết lập chung, nhập mô-đun, đặt thuộc tính môi trường và kiểm tra tiện ích mở rộng phân tích không gian, nằm ngoài vòng lặp, trước khi bắt đầu lặp lại. Các câu lệnh này chỉ được thực hiện một lần; không cần phải lặp lại chúng. Các bước xử lý địa lý lặp lại đặt tên, tạo và lưu raster đầu ra được đặt bên trong vòng lặp nơi chúng thuộc về.

**Ví dụ 10.2: Sử dụng while-loop để tạo 3 raster chứa các giá trị ngẫu nhiên có phân phối chuẩn.**

  
***10.1.2 FOR-Loops***

**FOR**-loops rất cần thiết để xử lý dữ liệu trong Python. Giả sử bạn có một danh sách các tệp. Bạn có thể thực hiện xử lý hàng loạt bằng cách sử dụng danh sách để kiểm soát vòng lặp. Trong ví dụ 10.3, 'simpleForLoop.py' in tên của từng tệp trong danh sách.

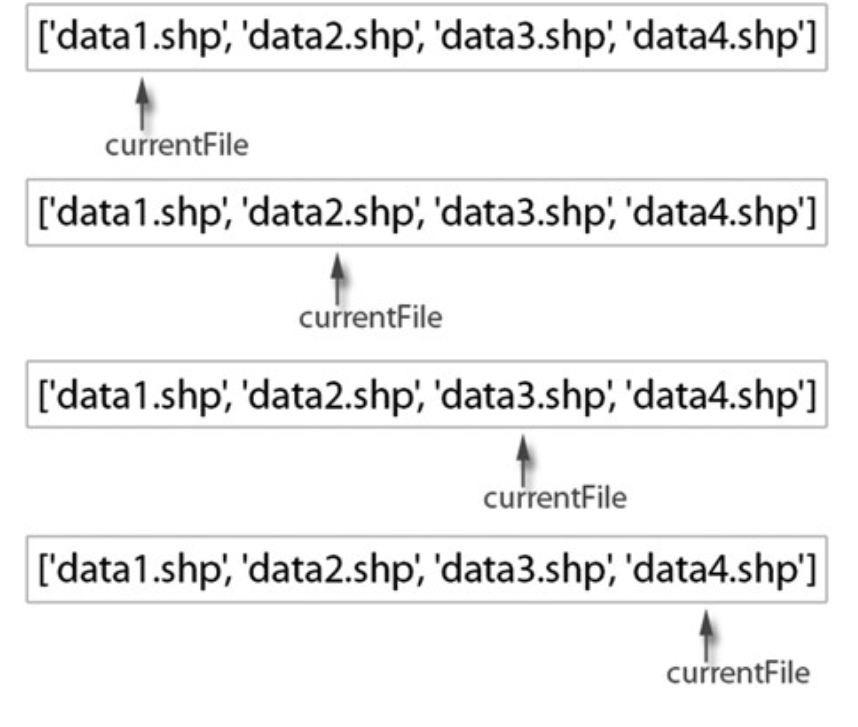
**Ví dụ 10.3: Basic FOR-loop.**



Khi bạn lần đầu tiên nhìn thấy vong lặp for trong Python, nó có thể xuất hiện bí ẩn vì các cơ chế tạo ra vòng lặp xảy ra không rõ ràng. Biến được đặt giữa for và in từ khóa tự động lặp lại qua các mục trong danh sách theo sau trong từ khóa. Nói cách khác, nó giả định giá trị của từng mặt hàng kế tiếp nhau. Để hiểu những gì đang xảy ra trong Ví dụ 10.3, nó có thể hữu ích để hình dung một mũi tên được gắn nhãn **currentFile** di chuyển qua danh sách, như thể hiện trong Hình 10.1. **currentFile** lặp lại các mục trong **dFiles** danh sách. Nó giả định giá trị của từng mặt hàng kế tiếp nhau. Mỗi khi nó giả định một giá trị mới, khối mã thụt lề sẽ được thực thi (khối mã thụt lề chỉ bao gồm dòng 4 trong Ví dụ 10.3). Khi không còn giá trị nào chưa sử dụng trong danh sách, quá trình lặp hoàn tất và dòng mã được suy ra tiếp theo được thực thi (dòng 5 trong Ví dụ 10.3).

Tóm tắt nội dung thảo luận về Ví dụ 10.3, Vòng lặp for trong Python lặp lại khối mã thụt lề nhiều lần khi có các mục trong danh sách. Ví dụ 10.3 thực thi **print currentFile** bốn lần. Các từ khóa Python **for** và **in** được ghép nối với nhau. Bất cứ khi nào bạn sử dụng **for** từ khóa, bạn phải sử dụng **in** từ khóa cùng với nó. Cú pháp cho Python FOR-loops như sau:

for iteratingVar in sequence:  
 code statement(s)

**Hình 10.1** Python tự động cập nhật biến,   
**currentFile**trong ví dụ 10.3, cho từng mục trong danh sách.

Như với **WHILE**-loop, dấu hai chấm và khối mã thụt lề là cú pháp bắt buộc. Các biến lặp, **iteratingVar**, được gán các giá trị liên tiếp trong dãy. Các kiểu dữ liệu Python bao gồm một tập hợp các mục là các kiểu dữ liệu trình tự. Ví dụ bao gồm chuỗi Python, bộ dữ liệu và danh sách. Các chuỗi được làm bằng các ký tự; Các bộ dữ liệu và danh sách được tạo thành từ các mục được phân tách bằng dấu phẩy. Đối tượng dữ liệu trình tự mà chúng tôi sử dụng thường xuyên nhất là danh sách Python, mặc dù các loại khác như chuỗi Python và bộ dữ liệu cũng có thể được đặt **in** từ khóa.

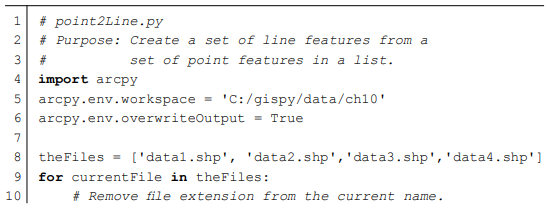
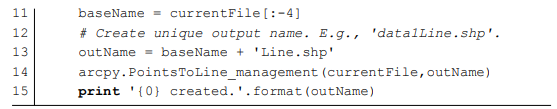
Thí dụ 10.3 đã sử dụng một biến có tên **currentFile** là biến lặp, nhưng không có gì đặc biệt về tên này. Mặc dù bạn có thể sử dụng bất kỳ tên biến nào, nhưng thông thường bạn không muốn sử dụng tên biến đã được sử dụng cho một số mục đích khác vì điều này sẽ thay đổi giá trị của biến đó. Bất kỳ tên biến nào bạn đặt giữa **for** và **in** từ khóa tự động được gán từng giá trị trong danh sách khi vòng lặp lặp lại. Giả sử bạn thích elephant. Đoạn mã sau cho thấy rằng bạn có thể đặt tên cho biến lặp của mình **elephant** (mặc dù chúng tôi thường chọn một thuật ngữ biểu thị nhiều hơn các mục trong danh sách):  
>>> dFiles = ['data1.shp', 'data2.shp', 'data3.shp', 'data4.shp']  
>>> **for** elephant **in** dFiles:  
. . . **print** elephant  
. . .  
data1.shp  
data2.shp  
data3.shp  
data4.shp

Lưu ý dấu ba chấm trong mẫu mã Cửa sổ tương tác PythonWin ở trên. Các dấu chấm này được sử dụng trong Cửa sổ tương tác để củng cố rằng dòng mã hiện tại là một phần của khối mã. Chúng tự động xuất hiện khi một khối mã (sau dấu hai chấm) được nhập vào cửa sổ tương tác. Trên thực tế, khi bạn nhập dòng cuối cùng của khối mã này, bạn phải nhấp lại vào phím 'Enter' để thoát khỏi khối mã vòng lặp. Chỉ khi đó mã mới được thực thi.  
 Thí dụ 10.4 Cho thấy một **FOR**-loop về xử lý địa lý vòng lặp với các số hàng ở bên trái. Tập lệnh này kết nối các điểm trong mỗi tệp đầu vào để tạo thành tệp đầu ra nhiều đường. Khi một tập lệnh tạo đầu ra bên trong một vòng lặp, tên đầu ra cần phải là duy nhất tại mỗi lần lặp; nếu không, một tệp có cùng tên sẽ được ghi qua mỗi lần và đầu ra cuối cùng sẽ là một tệp duy nhất. Để tạo một tên đầu ra duy nhất trong mỗi lần lặp lại của vòng lặp, bạn cần sử dụng một cái gì đó đang được cập nhật trong mỗi lần lặp. Trong ví dụ 10.4 **currentFile** biến đang được cập nhật ở mỗi lần lặp, vì vậy chúng ta cần sử dụng biến này trong tên tệp đầu ra.

|  |
| --- |
| Ghi chú nói chung, khi xử lý địa lý một tập hợp các tệp trong vòng lặp, biến lặp được sử dụng để xây dựng tên tệp đầu ra, vì biến này luôn thay đổi. |

Trong ví dụ 10.4, công cụ Điểm đến Dòng (Quản lý Dữ liệu) được gọi trên dòng 14. Các dòng khác bên trong khối mã liên quan đến việc tạo tên tệp đầu ra duy nhất. Để làm như vậy, họ sử dụng tên của tệp đầu vào như một phần của tên cho tệp đầu ra. Dòng 11 lấy tên cơ sở của đầu vào (data1, data2, …). Dòng 13 thêm 'Line.shp' để tạo các tên như 'data1Line.shp', 'data2Line.shp', v.v. Đây là một ví dụ cơ bản về xử lý địa lý trong vòng lặp. Các biến môi trường chỉ cần được đặt một lần, vì vậy chúng được đặt trước vòng lặp (giống như trong Ví dụ10,2). Trong ví dụ 10.4, một danh sách phải được khởi tạo trước vòng lặp, vì điều này được sử dụng để tạo vòng lặp. Tiếp theo là câu lệnh lặp theo sau là khối mã thụt lề. Mã này thiết lập tên biến đầu ra và sau đó thực hiện xử lý địa lý.

**Ví dụ 10.4: Sử dụng FOR-loop** **để thực hiện chuyển đổi điểm thành dòng trên danh sách tệp được mã hóa cứng.**



Các ví dụ 10.1-10.4, hiển thị lặp lại dựa trên số và lặp lại dựa trên danh sách. Ở đó là những cách sử dụng khác của những cấu trúc này, nhưng đây là những cách phổ biến nhất. Các ví dụ được sử dụng **WHILE**-loops để lặp lại dựa trên số và **FOR**-loops để lặp lại dựa trên danh sách. Trên thực tế, **FOR**-loop cũng có thể thực hiện lặp lại dựa trên số bằng cách sử dụng danh sách số. Thay vì mã hóa một danh sách các số, phần mềm tích hợp sẵn hàm **range**, trả về danh sách số, có thể được sử dụng để tự động tạo danh sách số. Ví dụ: đoạn mã sau trả về một danh sách Python chứa các số nguyên từ 1 đến 5:  
>>> range (1,6)  
[1, 2, 3, 4, 5]  
Bạn có thể đặt một biến thành giá trị trả về hoặc nhúng phạm vi hoạt động trực tiếp trong **FOR**-loop câu lệnh lặp, như sau.  
>>> **for** i in range(1,6):

… **print** i

1

2

3

4

5

Thí dụ 10,5 sử dụng **range** chức năng trong một **FOR**-loop để đệm dữ  
liệu. Các số từ một đến năm được sử dụng để thay đổi khoảng cách đệm từ  
một đến năm dặm. Các số cũng được sử dụng để tạo tên đầu ra. Biến lặp tạo  
thành phần duy nhất của tên đầu ra.

**Ví dụ 10.5: FOR-loop bằng cách sử dụng phạm vi chức năng cập nhật đơn vị tuyến tính**.

# bufferLoopRange.py  
# Mục đích: Tạo vùng đệm cho một công viên có khoảng cách đệm khác nhau # từ 1 đến 5 dặm.  
**import** arcpy  
arcpy.env.workspace = 'C:/gispy/data/ch10'  
arcpy.env.overwriteOutput = True  
inName = 'park.shp'  
**for** num **in** range(1, 6):  
 # Set buffer distance based on num ('1 miles', '2 miles', ...)  
 distance = '{0} miles'.format(num)  
 # Set output name ('buffer1.shp', 'buffer2.shp', ...)

outName = 'buffer{0}.shp'.format(num)  
 arcpy.Buffer\_analysis(inName, outName, distance)  
 print '{0} created.'.format(outName)

**10.2 Khối mã lồng nhau**

Các tập lệnh thường cần sử dụng các cấu trúc mã lồng nhau — chẳng hạn như các khối điều kiện bên trong **FOR**-loops (Ví dụ 10.6) hoặc **FOR**-loops inside **FOR**-loops (Ví dụ 10.7). Lưu ý cách thụt lề vào trong khi lồng xảy ra. Trong ví dụ 10.6 **if**, **elif**, và **else** từ khóa được căn chỉnh ở một cấp và **print** các từ khóa chứa trong các khối này được căn chỉnh ở cấp độ tiếp theo. Tập lệnh 'emotaLoop.py' lấy tên từ các biểu tượng cảm xúc mà nó in ra. Tập lệnh này lặp qua từng tên tệp trong danh sách và xác định mặt nào sẽ in, dựa trên phần mở rộng tệp. **End-switch** cùng phương thức trả về **True** hoặc **False**, được sử dụng để kiểm tra phần mở rộng của tệp.

Ví dụ 10.6

# emotaLoop.py  
# Mục đích: Các điều kiện lồng nhau bên trong vòng lặp để in biểu tượng cảm xúc  
# cho mỗi tên tệp.  
theFiles = ['crops.csv', 'data1.shp', 'rast','xy1.txt']

for f in theFiles:  
 if f.endswith('.shp'):  
 print ' ;] ' + f  
 elif f.endswith('.txt'):  
 print ' :( ' + f  
 else:  
 print ' :o ' + f

Đầu ra đã in:  
:o crops.csv  
;] data1.shp  
:o rast  
:( xy1.txt

Thí dụ 10.7, 'Scting.py', sử dụng các vòng lặp lồng nhau để in một mẫu nhịp điệu. Có thể bạn dự đoán đầu ra bằng cách đọc mã? Dấu cách được nhúng trong các câu lệnh in để phản ánh mức độ lồng vào nhau. Những người bên ngoài **FOR**-loops không có khoảng trắng. Những người bên trong đầu **FOR**-loop là bốn dấu cách. Những người bên trong lồng nhau **FOR**-loop được tiếp tục bởi tám dấu cách. Với khoảng cách này được chèn trong các câu lệnh in, thụt lề đầu ra bắt chước thụt lề của các dòng mã đã tạo ra nó.

Ví dụ 10.7  
# scatting.py  
# Purpose: Use nested loops to scat.  
print '\nskeep-de'  
for i in range(2):  
 print ' beep'  
 for j in range(3):  
 print ' bop'  
print 'ba-doop!'

Đầu ra đã in:  
skeep-de  
 beep  
 bop  
 bop  
 bop  
 beep  
 bop  
 bop  
 bop  
ba-doop!

**10.3 Kiểm kê Thư mục**

**FOR**-loops có thể được sử dụng với Python tích hợp sẵn hệ điều hành chức năng mô-đun **listdir** và **walk** để điều hướng dữ liệu và thư mục. Chúng tôi chứng minh sự đơn giản **listdir** chức năng ở đây và phức tạp hơn **os.walk** chức năng trong Chương 12. Các **listdir** chức năng liệt kê các tệp trong thư mục đầu vào, chẳng hạn như thư mục dữ liệu mẫu 'ảnh'. Đoạn mã sau in ra danh sách các tệp trong 'C: / gispy / data / ch10 / pics':

>>> **import** os  
>>> theDir = 'C: / gispy / data / ch10 / pics'  
>>> # os.listdir trả về một danh sách các tệp  
>>> theFiles = os.listdir (theDir)  
>>> theFiles  
['istanbul.jpg', 'istanbul2.jpg', 'italy', 'Jerusalem', 'marbleRoad. jpg ',' Cay\_market.jpg ', 'stage.jpg ']

Đầu ra là một danh sách Python, vì vậy for-loop có thể được sử dụng để lặp qua các tệp như sau:  
>>> for fileName in theFiles:  
. . . **print** fileName  
. . .  
istanbul.jpg  
istanbul2.jpg  
italy  
Jerusalem  
marbleRoad.jpg  
Cay\_market.jpg  
stage.jpg

Thí dụ 10,8 liệt kê các tệp trong 'C: / gispy / data / ch10' và tạo bản sao của bất kỳ tệp văn bản nào trong thư mục. Không gian làm việc xử lý địa lý được đặt và sau đó giá trị này được chuyển vào hàm **listdir**. Chỉ một **arcpy** chức năng mô-đun bị ảnh hưởng khi **arcpy.env.workspace** được thiết lập. Các hệ điều hành mô-đun không xem xét cài đặt môi trường này, vì vậy giá trị này vẫn phải được chuyển cho **listdir** hàm số. 'copyLoop.py' sử dụng chuỗi tận cùng để tìm các tệp có phần mở rộng 'txt', nhưng thủ thuật này không phải lúc nào cũng hoạt động. Ví dụ: giả sử bạn muốn sao chép bất kỳ tệp nào là raster. Có thể khó kiểm tra tất cả các tiện ích mở rộng. Một số thậm chí có thể không có phần mở rộng, chẳng hạn như tệp raster Esri GRID. Trên thực tế, nhu cầu xử lý một loại tệp cụ thể rất phổ biến trong quy trình làm việc GIS đến mức **arcpy** gói có các chức năng để nhận danh sách các kiểu dữ liệu cụ thể trong một thư mục. Các chức năng này được giới thiệu trong chương tiếp theo.

**Ví dụ 10.8: Liệt kê các tệp với hệ điều hành mô-đun và xử lý địa lý các tệp.**  
# copyLoop.py  
# Mục đích: Tạo một bản sao của mỗi tệp phần mở rộng ASCII .txt.  
**import** arcpy, os  
arcpy.env.workspace = 'C:/gispy/data/ch10'  
outDir = 'C:/gispy/scratch/'  
theFiles = os.listdir(arcpy.env.workspace)  
for fileName in theFiles:  
 if fileName.endswith('.txt'):  
 outName = outDir + fileName[:-4] + 'V2.txt'  
 arcpy.Copy\_management(fileName, outName)  
 **print** '{0} created.'.format(outName)

Lưu ý rằng listdir hàm trả về một danh sách các tên cơ sở, không phải tên tệp đường dẫn đầy đủ. Nhưng os.path các phương thức trả về thông tin tệp, chẳng hạn như kích thước hoặc ngày sửa đổi, cần biết vị trí của tệp. Nếu chỉ có một tên cơ sở được đưa ra, phương thức sẽ tìm kiếm tệp trong hệ điều hành thư mục làm việc hiện tại của module. Nếu nó không được tìm thấy ở đó, nó sẽ không thể tìm thấy tệp. Ví dụ, os.path.exists hàm trả về thật nếu nó xác định rằng tệp được truyền dưới dạng đối số tồn tại. Đoạn mã sau liệt kê các tệp trong một thư mục, sau đó kiểm tra xem chúng có tồn tại hay không:

>>> **import** os  
>>> theDir = 'C: / gispy / data / ch10 / pics'  
>>> # os.listdir trả về một danh sách các tệp  
>>> theFiles = os.listdir (theDir)  
>>> **for** fileName **in** theFiles:  
. . . **print** os.path.exists (fileName)  
. . .  
False  
False  
False

. . . (và kể từ đó trở đi)

Các tệp không bị xóa giữa khi liệt kê chúng và kiểm tra sự tồn tại. Thay vào đó, các tệp không được tìm thấy bởi os.path.exists vì đường dẫn đầy đủ không được chỉ định. Để chỉ định đường dẫn đầy đủ, hãy nối thư mục và tên tệp:

>>> fullName = os.path.join (theDir, fileName)  
>>> os.path.exists (fullName)  
True

Thí dụ 10.9 sử dụng **os.path.getmtime** để in dấu thời gian cho lần cuối cùng một tệp được sửa đổi. Các **os.path.join** phương thức được sử dụng bên trong vòng lặp để tạo tên tệp đường dẫn đầy đủ cho các tệp. Thí dụ 10.8 bộ **arcpy.env.workspace** làm việc vì vậy mà **arcpy.Copy** có thể xác định vị trí các tệp. Nhưng **arcpy** không gian làm việc không ảnh hưởng đến hệ điều hành thư mục làm việc hiện tại của module. Đây là lý do tại sao bạn cần sử dụng tên tệp đường dẫn đầy đủ trong Ví dụ 10.9, nhưng không phải trong Ví dụ 10.8. Nó cũng có thể thay đổi hệ điều hành thư mục làm việc hiện tại của mô-đun, sử dụng **os.chdir** để chỉ huy. Nhưng điều quan trọng là phải nhận ra rằng, điều này không ảnh hưởng đến **arcpy.env.workspace** thiết lập.

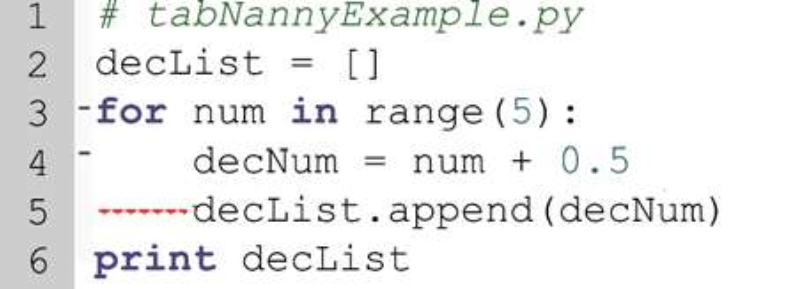
**Ví dụ 10.9: Sử dụng os.path.join bên trong một vòng lặp để tạo tên tệp đường dẫn đầy đủ.**# printModTime.py  
# Mục đích: Đối với mỗi tệp, in thời gian nhiều nhất  
#  
# Đầu vào: sửa đổi gần đây.Không cần đối số.  
**import** os, datetime

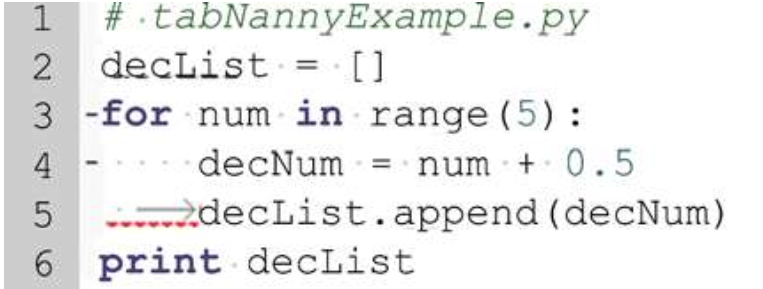
theDir = 'C: / gispy / data / ch10 / pics'  
theFiles = os.listdir (theDir)

for f in theFiles:  
 fullName = os.path.join(theDir, f)  
 # Get the modification time.  
 print os.path.getmtime(fullName)

**10.4 Thụt lề và TabNanny**

Chương 4 đã giới thiệu nút kiểm tra cú pháp PythonWin cú pháp và chạy cả TabNanny. Bây giờ bạn đang viết các khối mã thụt lề, hãy truy cập lại TabNanny, tính năng này sẽ kiểm tra khoảng cách phù hợp. TabNanny sẽ chèn các dấu gạch chân trước mã thụt lề nếu tìm thấy sự mâu thuẫn trong thụt lề. Bây giờ bạn biết rằng thụt lề trong một khối cần phải được căn chỉnh. Các dấu TabNanny chỉ ra một vấn đề phức tạp hơn đôi khi xảy ra khi mã được sao chép từ các nguồn như tệp PDF hoặc Microsoft® tài liệu phần mềm sang PythonWin. Ví dụ: mã sau đây đã được sao chép từ PowerPoint® trình bày với PythonWin:

  
  
Để hiểu nguyên nhân của lỗi, hãy hiển thị 'Khoảng trắng' trong PythonWin (Chế độ xem> Khoảng trắng). Khoảng trắng bao gồm các ký tự ẩn (chẳng hạn như dấu cách đơn lẻ và tab) được sử dụng để thụt lề và phân tách các phần tử trong một dòng mã. PythonWin hiển thị khoảng trắng dưới dạng chấm màu xám và các tab dưới dạng mũi tên. Đây là cách ví dụ của chúng tôi trông với Khoảng trắng được bật:



Sự cố xảy ra do việc sử dụng Khoảng trắng không nhất quán; Dòng 4 sử dụngbốn dấu cách và dòng 5 sử dụng một dấu cách và một tab. Thay thế tab trên dòng 5 bằng ba dấu cách để sửa chữa sự cố.

**10.6 Bài tập**

1. **whileLoopino.py** Viết lại tập lệnh sau, thay thế **FOR**-loop với **WHILE**-loop.

For n in range(-50,150,5):  
 print n

Lưu ý: Dấu phẩy ở cuối câu lệnh in KHÔNG phải là lỗi đánh máy. Đó là một cách để làm cho các câu lệnh in liên tiếp xuất hiện trên cùng một dòng được phân tách bằng dấu cách.

2. **forLoopino.py** Viết lại tập lệnh sau, thay thế **WHILE**-loop với một  
**FOR**-loop và sử dụng phạm vi hàm số.  
  
 index = 9  
 while index <= 99:  
 print index  
 index = index + 10

3. **outline.py** Viết một tập lệnh sử dụng lồng nhau **FOR**-loops để in đầu ra được hiển thị bên dưới. Sử dụng cài sẵn phạm vi hàm cho vòng lặp số (bên ngoài) và sử dụng danh sách được mã hóa cứng ['a', 'b', 'c'] cho vòng lặp bên trong. Bắt đầu các câu lệnh in vòng lặp bên trong bằng một tab để thụt lề chúng như được hiển thị.

1) Hehe  
a) Hoho  
b) Hoho  
c) Hoho

2) Hehe  
a) Hoho  
b) Hoho  
c) Hoho

3) Hehe  
a) Hoho  
b) Hoho  
c) Hoho

4) Hehe  
a) Hoho  
b) Hoho  
c) Hoho

4. **loopDistance.py** Viết tập lệnh để tìm một tập hợp các bảng khoảng cách từ trạm cứu hỏa đến trường học trong các lớp tính năng trong 'C:/gispy/data/ch10/county.gdb'. Sử dụng công cụ Khoảng cách điểm (Phân tích) để xác định khoảng cách từ các trạm cứu hỏa (đối tượng điểm đầu vào) đến trường học (đối tượng địa lý gần) và sử dụng **WHILE**-loop để chạy công cụ với mười giá trị bán kính tìm kiếm khác nhau: 500 mét, 1000 mét,…, 5000 mét. Sử dụng 'C: / gispy / xước' làm thư mục đầu ra cho mười bảng đầu ra, thay vì tạo chúng bên trong cơ sở dữ liệu địa lý tệp. Đặt tên cho các tệp đầu ra bằng khoảng cách số là 'dist500.dbf', 'dist1000.dbf', 'dist1500.dbf', v.v. Mã cứng các tính năng đầu vào, các tính năng gần và thư mục đầu ra trong tập lệnh này để không cần đối số tập lệnh.

5. **loopAggregate.py** Để so sánh kết quả của việc sử dụng công cụ Đa giác tổng hợp (Bản đồ) với các giá trị khác nhau cho tham số khoảng cách tổng hợp, hãy viết tập lệnh tổng hợp các đa giác trong 'park.shp' cho mười khoảng cách tổng hợp khác nhau: 150 thước, 250 thước,…, 1050 thước . Sử dụng cài sẵn phạm vi chức năng và một **FOR**-loop. Thay đổi tên đầu ra dựa trên trình lặp vòng lặp ('park150\_agg.shp', 'park250\_agg.shp', v.v.). Giả sử 'park.shp' nằm trong 'C: / gispy / data / ch10 /' và đặt đầu ra trong 'C: / gispy / xước'. Không cần đối số kịch bản.

6. **loopSimplify.py** Để khảo sát kết quả của việc sử dụng công cụ Đơn giản hóa Đường (Bản đồ) với các giá trị dung sai khác nhau, hãy viết một tập lệnh đơn giản hóa các đường trong 'parkLines.shp' với bốn giá trị dung sai khác nhau. Sử dụng cài sẵn phạm vi chức năng và một **FOR**-loop để thực hiện đơn giản hóa đường đối với các giá trị dung sai: 30 feet, 60 feet, 90 feet và 120 feet. Sử dụng thuật toán 'POINT\_REMOVE'. Đặt tên đầu ra là 'simp30.shp', 'simp60.shp', 'simp90.shp' và 'simp120. shp ', và đặt đầu ra trong' C: / gispy / xước '. Không cần đối số kịch bản.

7. **loopGetCount.py** Sử dụng **os.listdir** để lấy danh sách các tệp hình dạng trong 'C: / gispy / data / ch10 /'. Sau đó, đối với bất kỳ tệp hình dạng nào có tên chứa từ 'out', không phân biệt chữ hoa chữ thường, hãy sử dụng công cụ Get Count (Data Management) để xác định số lượng bản ghi trong bảng thuộc tính. Sử dụng một đối số script, một đường dẫn thư mục. Báo cáo kết quả như trong ví dụ.  
Ví dụ đầu vào: C: / gispy / data / ch10 /archive

Đầu ra ví dụ:  
linesOUT.shp has 530 entries.  
outData.shp has 100 entries.  
parkOutput.shp has 426 entries.

8. **loopFileSize.py** Sử dụng **os.listdir** để lấy danh sách các tệp trong một thư mục. Sau đó, sử dụng **os.path.getsize** chức năng in tên của các tệp nhỏ và kích thước của chúng. Cho phép người dùng chỉ định kích thước tối đa tính bằng byte. Sử dụng hai đối số tập lệnh, đường dẫn thư mục và ngưỡng kích thước tính bằng byte. Báo cáo kết quả như trong ví dụ.  
Ví dụ đầu vào: C: / gispy / data / ch10 / 64

Đầu ra ví dụ:  
data.txt is 42 bytes.  
xy1.txt is 42 bytes.  
xy\_current.txt is 44 bytes.