**5 STRUCTURED TYPES, MUTABILITY, AND HIGHER-ORDER FUNCTIONS**

الانواع المركبة – الأنواع القابلة للتغير – الدوال عالية الرتبة

حتى الان، كل أنواع البرامج التى عملنا عليها كانت تحتوى على ثلاثة انواع أساسية من البيانات ، الأعداد الطبيعية INT والأعداد التى تحتوى على نقطة عائمة FLOAT والسلاسل الحرفية ( النصوص ) STR ، وكلها بيانات من النوع المفرد (Scalar) ، ولهذا يمكننا أن نقول أنها كائنات لا نستطيع الوصول الى مكوناتها لانها لا تحتوى على مكونات ، كل كائن منها عبارة عن وحدة واحدة .

على عكس النصوص (str) يمكننا أن نراها على أنها مركبة ، أو نوع من البيانات الغير فردية ، لانك قد تستخدم الفهرسة (indexing) لاستخراج حرف أو أكثر عن طريق تشريح النص وتكوين نصوص فرعية منه.

فى هذا الفصل ، سوف نتعرض الى ثلاثة أنواع جديدة للبيانات يمكننا أن نقول عنها متراكبة ، أى تتكون من أجزاء ، أولها يسمى المجموعة (tuble) وهو بالاحرى تعميم بسيط للنوع المعروف str . النوعين الآخرين أحدهما هى القائمة List والآخر هو الاملاءى (dict) ، وسوف تكون دراسها أكثر متعة باذن من الله. وذلك بسبب قابليتها للتغير.

سوف نعود أيضا للجزئية المتعلقة بالدوال ، مع بعض الامثلة التى تصور الادوات التى تتيح لنا التعامل مع الدوال مثلها مثل باقى انواع الكائنات.

# 5-1 المجموعاتTuples

مثل النصوص ، المجموعة تتكون من تتابع مرتب من العناصر. الفرق هنا هو ان عناصر المجموعة لا يشترط أن تكون حروف أو نصوص ، كل عنصر يمكن أن يكون أى نوع من البيانات ، ولا نحتاج الى ان تكون كل العناصر من نفس النوع ، بل ان كل عنصر يمكنه أن يكون نوع مستقل عن الاخر ، حرفيا تتم كتابة عناصر المجموعة كقائمة ويفصل بين كل عنصر وعنصر آخر بفاصلة ، ويحاط جميه العناصر بأقواس مدورة .على سبيل المثال يمكننا كتابة :

*t1 = ( )*

*t2 = (1, 'two', 3)*

*print t1*

*print t2*

ولا داعى للدهشة عندما ترى أن أمر الطباعة ينتج لك المخرجات التالية :

*( )*

*(1, 'two', 3)*

بالنظر الى هذا المثال ، ربما – و هذا شىء طبيعى – تعتقد ان المجموعة تحتوى على قيمة مفردة ، 1 سوف يتم كتابتها ( 1 ) ولكن ، على حد تعبير ريتشارد نيكسون (that would be wrong ) .

عندما ترى التعبير ( 1 ) ربما تعتقد ان هذه طريقة مطولة لكتابة العدد الصحيح 1 ، ولكنها تكتب هكذا للدلالة على مجموعة مفردات تحتوي على هذه القيمة ، نكتب ( 1, ) ، تقريبا كل شخص يستخدم بايثون لمرة أو أكثر قد قام بحذف - عن طريق الخطأ - تلك الفاصلة المزعجة ، مثل السلاسل النصية ، يمكن تجميع المجموعات ، فهرستها ، وتسريحها. انظر :

*t1 = (1, 'two', 3)*

*t2 = (t1, 3.25)*

*print t2*

*print (t1 + t2)*

*print (t1 + t2)[3]*

*print (t1 + t2)[2:5]*

التعيين الُثانى يربط المتغير t2 الى القيم التالية : المجموعة التى تم تعيينها الى المتغير t1 بالاضافة الى الرقم العشرى 3.25 ، وهذا جائز ، لأنه مثله مثل أى شىء آخر فى بايثون ، عبارة عن كائن ، لذا فالمجموعة يمكن أن تحتوى بداخلها على مجموعات ، تماما كما أخرجته لنا أمر الطباعة الأول :

*((1, 'two', 3), 3.25)*

أخرج لنا أمر الطباعة الثانى القيمة التى تم توليدها عن طريق الجمع بين القيمة التى تم تعيينها الى المتغير t1 مضافا اليها القيمة التى تم تعيينها الى المتغير t2 وقامت باخراج الناتج:

*(1, 'two', 3, (1, 'two', 3), 3.25)*

امر الطباعة التالى قام بانتقاء وطباعة العنصر الرابع الناتج من جمع المجموعات ( دائما ما تبدأ الفهارس فى بايثون من الصفر ) ، وقامم أمر الطباعة التالى بانشاء وطباعة شريحة من المجموعة كما فى المخرجات التالية :

*(1, 'two', 3)*

*(3, (1, 'two', 3), 3.25)*

يمكننا استخدام الحلقة التكرارية for لتكرر نفسها وتلتقط عناصر المجموعة ، على سبيل المثال الكود التالى يقوم بحساب القواسم المشتركة للأعداد 20 و 100 ثم يقوم بعدها بجمع هذه القواسم :

*def findDivisors (n1, n2):*

*""" Assumes that n1 and n2 are positive ints  
 Returns a tuple containing all common divisors of n1 & n2"""  
 divisors = () #the empty tuple  
 for i in range(1, min (n1, n2) + 1):  
 if n1%i == 0 and n2%i == 0:* *divisors = divisors + (i,)  
 return divisors*

*divisors = findDivisors(20, 100)  
 print divisors total = 0  
 for d in divisors:  
 total += d  
 print total*

# 5-1-1 التتالى والتعيين المتعدد Sequences and Multiple Assignment

اذا كنت تعرف عدد مكونات سياق معين – على سبيل المثال سلسلة حرفية أو مجموعة – انه الوقت المناسب لاستخدام التعيين المتعدد فى بايثون لاستخراج عناصر مفردة ، على سبيل المثال ، بعد تنفيذ حالة التعيين x , y = ( 3, 4 ) سوف يتم تعيين القيمة 3 الى x وتعيين القيمة 4 الى y ، وبالمثل حالة التعيين a, b, c = ‘xyz’ سوف يتم تعيين القيمة ‘x’ الى المتغير a والقيمة ‘y’ الى المتغير b والقيمة ‘z’ الى المتغير c . هذه الآلية تناسب الدوال التى تعيد سياقات محددة الحجم.

*Consider, for example the function*

*def findExtremeDivisors(n1, n2):*

*""" Assumes that n1 and n2 are positive ints*

*Returns a tuple containing the smallest common divisor > 1 and the*

*largest common divisor of n1 and n2 """*

*divisors = () #the empty tuple*

*minVal, maxVal = None, None*

*for i in range(2, min(n1, n2) + 1):*

*if n1%i == 0 and n2%i == 0:*

*if minVal == None or i < minVal:*

*minVal = i*

*if maxVal == None or i > maxVal:*

*maxVal = i*

*return (minVal, maxVal)*

*The multiple assignment statement*

*minDivisor, maxDivisor = findExtremeDivisors(100, 200) will bind*

*minDivisor to 2 and maxDivisor to 100.*

# 5-2 القوائم والقابلية للتغير Lists and Mutability

مثل المجموعات ، القواءم Lists هى سياق مرتب من القيم ، عندما تكون كل قيمة معرفة بفهرس . القاعدة اللغوية للتعبير عن عناصر القائمة يشبه تلك المستخدمة فى المجموعات ؛ الفرق هو أننا نستخدم أقواسا مربعة بدلا من تلك الدائرية التى نستخدمها مع المجموعات ، والقائمة الفارغة تكتب [ ] ، والقائمة المفردة بدون تلك الفاصلة قبل اغلاق القوس ، لهذا ، على سبيل المثال :

*L = ['I did it all', 4, 'love']*

*for i in range(len(L)):*

*print L[i]*

سوف تكون النتيجة :

*I did it all*

*4*

*love*

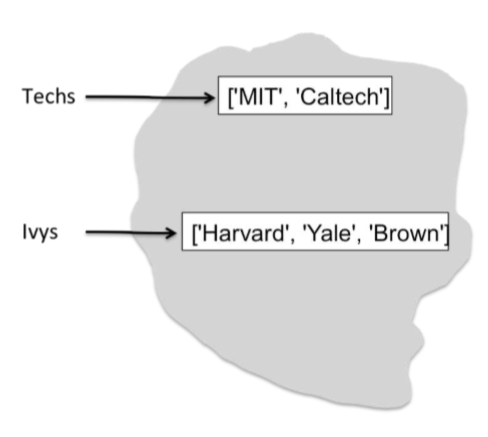
بالمناسبة ، حقيقة استخدام الاقواس المربعة هى لأنه حرفيا نوع البيانات – القواءم Lists – مفهرسة الى عناصر ، وتشريح القوائم ربما يؤدى الى بعض التداخل أو التعارض ، على سبيل المثال ، التعبير [1, 2, 3, 4][1:3][1] ، قيمته 3 يستخدم الاقواس المربعة بثلاث طرق مختلفة ، هذه نادرا ما يكون بسبب مشكلة فى الممارسة ، لانه فى معظم الوقت يتم بناء القوائم عن طريق الاطراد وليس حرفيا كما يحدث فى المجموعات ، وتختلف القوائم عن المجموعات فى أن الأولى قابلة للتغيرMutable . في المقابل ، المجموعات و السلاسل الحرفية غير قابلة للتغيير Immutable . هناك العديد من العوامل التي يمكن استخدامها لإنشاء كائنات من هذه الأنواع الغير قابلة للتغيير . والمتغيرات يمكن ربطها بكائنات من هذه الأنواع . لكن الكائنات من النوع Immutable لا يمكن التعديل عليها . بينما على الجانب الآخر فان الكائنات من النوع List يمكن التعديل عليها بعد انشائها ، التمييز بين عملية تغير كائن وعملية تعيين كائن الى متغير ربما – للوهلة الاولى – نراها خفية . على كل حال يمكنك الاستمرار فى تلاوة التعويذة .

فى بايثون المتغير هو مجرد اسم ،ن على سبيل المثال – علامة يتم الحاقها بكائن ما ، ربما عملت التعويذة عملها وظهر جزء من الغموض ، لنرى هذه السطور ربما تتضح الامور أكثر :

*Techs = ['MIT', 'Caltech']*

*Ivys = ['Harvard', 'Yale', 'Brown']*

عند التنفيذ ، سيقوم المترجم Interpreter بانشاء قائمتين ، وربط كل واحدة بالاسم المناسب لها كما فى الشكل التالى :



**الشكل 5 – 1 قائمتين**

ولنلاحظأيضا فى عمليات التعيين الآتية :

*Univs = [Techs, Ivys]*

*Univs1 = [['MIT', 'Caltech'], ['Harvard', 'Yale', 'Brown']]*

هنا أيضا يقوم المترجم بانشاء قائمتين جديدتين وبربط لهما المتغيرات كما يبدو لنا . وعناصر هذه القواءم هى نفسها قوائم ، لنأخذ أوامر الطباعة الثلاثة الآتية :

*print 'Univs =', Univs*

*print 'Univs1 =', Univs1*

*print Univs == Univs1*

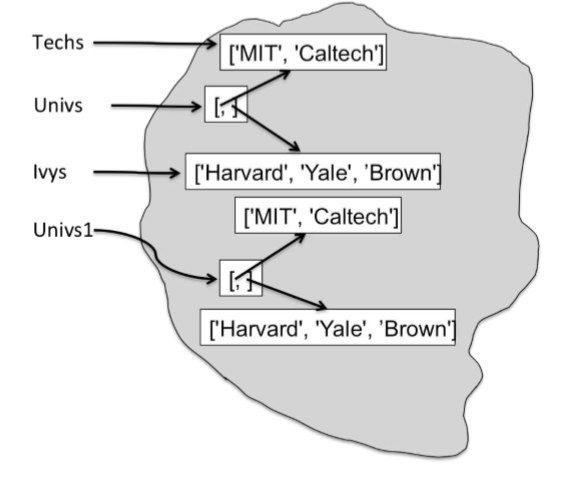
وشيكون الناتج :

*Univs = [['MIT', 'Caltech'], ['Harvard', 'Yale', 'Brown']]*

*Univs1 = [['MIT', 'Caltech'], ['Harvard', 'Yale', 'Brown']]*

*True*

ما هذا ! ، يبدو كما لو أنهما قائمة واحدة ، ويبدو أيضا أن المتغيرات univ و univ1 قد تم ربطهما الى نفس القائمة ، لكن قد تبدو المظاهر خادعة فى بعض الاحيان ، وهذا ما سوف يتضح فى الشكل التالى أن المتغيرات univ و univ1 قد تم ربط كل منهما الى قيمة مغايرة تماما للاخرى :



**الشكل 5 – 2 صورة لقائمتين يبدو أنها واحدة لكنهما ليستا كذلك**

الصورة توضح أن المتغيرات univ و univ1 مربوطة الى قيم منفصلة تماما عن بعضها ، يمكننا التحقق من ذلك باستخدام الدالة الداخلية id والتى تعيد رقم تعريفى مختلف وفريد لكل كائن مخزن فى الذاكرة . وهذه الدالة عادة ما تستخدم لاختبار التكافؤ ، عندما يتم اجراء الكود لاتالى :

*print Univs == Univs1 #test value equality*

*print id(Univs) == id(Univs1) #test object equality*

*print 'Id of Univs =', id(Univs)*

*print 'Id of Univs1 =', id(Univs1)*

وسيكون ناتج الطباعة

*True*

*False*

*Id of Univs = 24499264*

*Id of Univs1 = 24500504*

لا تتوقع أن تحصل على نفس المعرفات عند اجراء هذا الكود ، الدلالات اللفظية فى بايثون semantics تقول لا تسأل عن ماهية معرفات الكائنات ولا كيف يتم توليدها ، هى فقط لمجرد تحقيق ان يكون لكل كائن رقم تعريفى فريد ،لاحظ فى الشكل 5 – 2 ان عناصر univ هى ليست نسخ من عناصر المجموعة المربوط اليها techs and lvys ولكن بالأحرى فانها القوائم نفسها ، العناصر فى univ1 هى قوائم تحتوى على نفس العناصر والتى تم سردها فى القائمة univ . لكنها ليست العناصر نفسها ، يمكننا التحقق من ذلك باجراء الكود :

*print 'Ids of Univs[0] and Univs[1]', id(Univs[0]), id(Univs[1])*

*print 'Ids of Univs1[0] and Univs1[1]', id(Univs1[0]), id(Univs1[1])*

وستكون النتيجة :

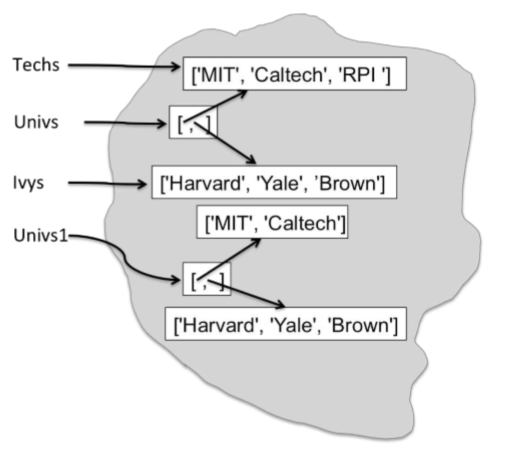
*Ids of Univs[0] and Univs[1] 22287944 22286464*

*Ids of Univs1[0] and Univs1[1] 22184184 22287984*

لماذا يحدث هذا؟! .. سبب هذا أن القوائم Lists هى كائنات قابلة للتغير ، وقد نلاحظ ذلك بعد اجراء :

Techs.append('RPI')

الأسلوب append له جانب تأثيرى ، فبدلا من انشاء قائمة جديدة ، هى تقوم بالتعديلفى القائمة الموجودة بالفعل Techs عن طريق اضافة عنصر جديد ، وسيتم اضافة النص ‘RPI’ بعد آخر العناصر الموجودة . وبعد الحاق العنصر ستكون حالى الحوسبة كالتالى :



**الشكل 5 – 3 الاثبات لعملية التحور**

القائمةuniv لازالت تحتوى على نفس القائمتين ، ولكن محتوى هاتين القائمتين قد تغيرتا ، فلنقم بإجراء أمر الطباعة التالى :

print 'Univs =', Univs

print 'Univs1 =', Univs1

وستكون نتيجة الطباعة :

Univs = [['MIT', 'Caltech', 'RPI'], ['Harvard', 'Yale', 'Brown']]

Univs1 = [['MIT', 'Caltech'], ['Harvard', 'Yale', 'Brown']]

ما حدث لدينا هنا هو شىء يسمى Aliasing – الاسم المستعار - ، يوجد هناك مساران متمايزان ، مسار واحد من خلال المتغير Tech والآخر من خلال العنصر الأول لكائن القائمة الذي يرتبط به Univ ، أحدهم أن يحور الكائن عبر أي من المسارين ، وسوف يكون تأثير التحور مرئياً من خلال كلا المسارين . قد يكون هذا مناسبا ، لكنه أيضا قد يكون غادرا . تؤدى الأسماء المستعارة الغير مقصودة الى أخطاء برمجية يكون من الصعب تعقبها .

كما فى المجموعات، الحلقة التكرارية for يمكنها أن تدور داخل القائمة وتخرج لنا محتواها ،

for e in Univ:

print 'Univ contains', e

print ' which contains'

for u in e:

print ' ', u

وسيكون الناتج :

Univs contains ['MIT', 'Caltech', 'RPI']

which contains

MIT

Caltech

RPI

Univs contains ['Harvard', 'Yale', 'Brown']

which contains

Harvard

Yale

Brown

عندما نضيف قائمة إلى أخرى ، على سبيل المثال ، ( Techs.append (Ivys) ) ، يتم الحفاظ على البنية الأصلية ، والنتيجة هي قائمة تحتوي على قائمة. لنفترض أننا لا نريد الحفاظ على هذا الهيكل ، ولكن نريد إضافة عناصر قائمة واحدة إلى قائمة أخرى. يمكننا القيام بذلك عن طريق استخدام تسلسل القوائم Concatenate أو طريقة التمديد Extend ، على سبيل المثال ،

L1 = [1,2,3]

L2 = [4,5,6]

L3 = L1 + L2

print 'L3 =', L3

L1.extend(L2)

print 'L1 =', L1 L1.append(L2)

print 'L1 =', L1

وسيكون الناتج :

L3 = [1, 2, 3, 4, 5, 6]

L1 = [1, 2, 3, 4, 5, 6]

L1 = [1, 2, 3, 4, 5, 6, [4, 5, 6]]

لاحظ أن المشغل + ليس له تأثير جانبي. يقوم بإنشاء قائمة جديدة وإرجاعها. في المقابل ، وتمديد وإلحاق كل تحور الى القائمة L1.

يحتوي الشكل 5.4 على أوصاف مختصرة لبعض الطرق المرتبطة بالقوائم. لاحظ أن كل هذه الأرقام باستثناء العد والفهرس تحوِّل القائمة

L.append(e) adds the object e to the end of L.

L.count(e) returns the number of times that e occurs in L.

L.insert(i, e) inserts the object e into L at index i.

L.extend(L1) adds the items in list L1 to the end of L.

L.remove(e) deletes the first occurrence of e from L.

L.index(e) returns the index of the first occurrence of e in L. It raises an exception (see Chapter 7) if e is not in L.

L.pop(i) removes and returns the item at index i in L. If i is omitted, it defaults to -1, to remove and return the last element of L.

L.sort() sorts the elements of L in ascending order.

L.reverse() reverses the order of the elements in L.

**5 – 2 – 1 النسخ Cloning**

على الرغم من أنه متاح ، لكن تجنب التتغيير فى قائمة تخضع للتكرار، لاحظ المثال التالى :

def removeDups(L1, L2):

"""Assumes that L1 and L2 are lists.

Removes any element from L1 that also occurs in L2"""

for e1 in L1:

if e1 in L2:

L1.remove(e1) L1 = [1,2,3,4] L2 = [1,2,5,6]

removeDups(L1, L2)

print 'L1 =', L1

ربما تتفاجأ عندما ترى أمر الطباعة يخرج هذه النتيجة :

L1 = [2, 3, 4]

خلال الحلقة for ، يقوم التطبيق Python Implementation بتتبع مكان وجوده في القائمة باستخدام عداد داخلي يتزايد في نهاية كل عملية تكرار. عندما تصل قيمة العداد إلى الطول الحالي للقائمة تتوقف الحلقة ، هذا قد يؤدى كما قد نتوقع منه أن يفعل إذا لم يتم تحور القائمة داخل الحلقة التكرارية ، ولكن يمكن أن يكون لها عواقب مفاجئة إذا تم تحوير القائمة ، في هذه الحالة ، يبدأ العداد المخفي عند 0 ، ويكتشف أن L1 [0] في L2 ، ويزيله - مما يقلل من طول L1 إلى 3. ثم يتم زيادة العداد إلى 1 ، وتستمر عملية للتحقق مما إذا كانت قيمةL1[1] موجودة في L2 . لاحظ أن القيمة الحالية لــ L1[1] ليست 2 ولكنها أصبحت 3 ، كما ترى أنه من السهل اكتشاف ما يحدث . . . تبدو أنها غير مقصودة كما فى هذا المثال .