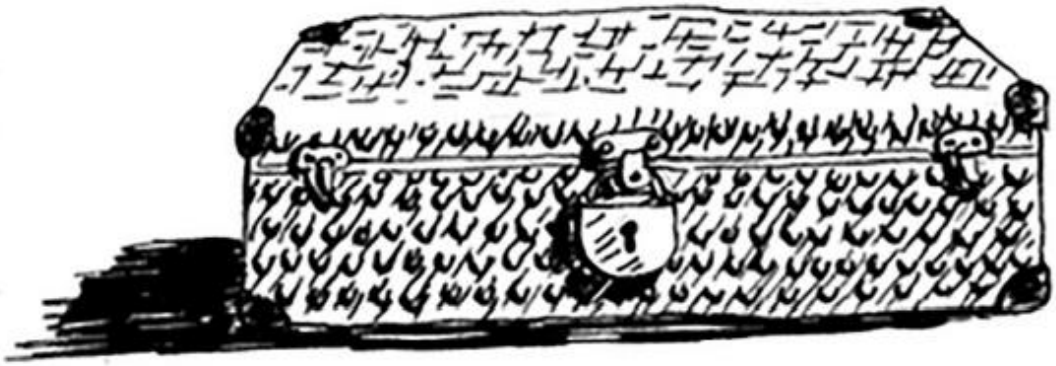
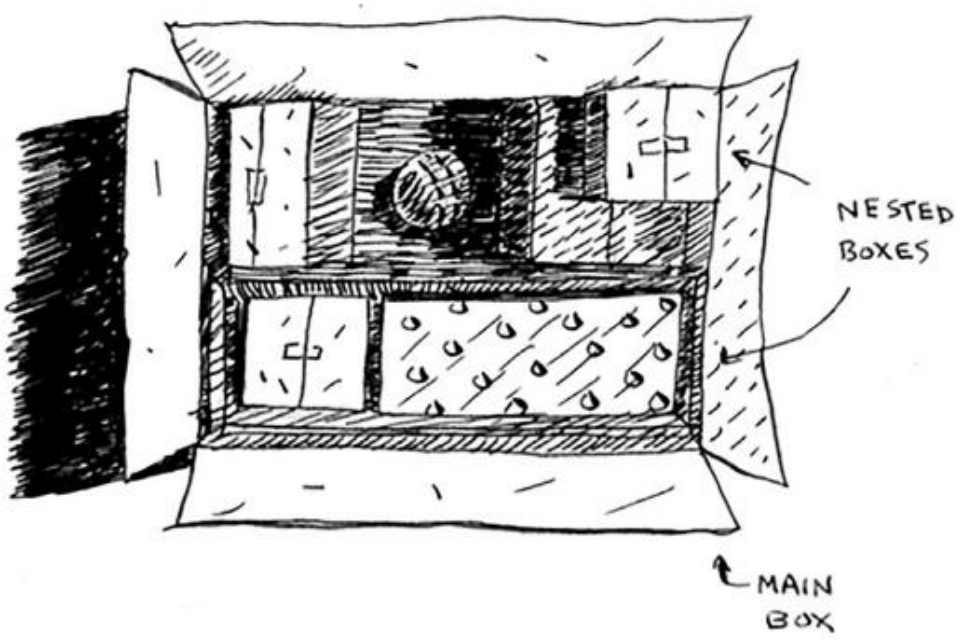
Recursion

(Chapter 3)

لنفترض انك تبحث في اغراض جدتك ووجدت حقيبة سفر غامضة مقفولة.

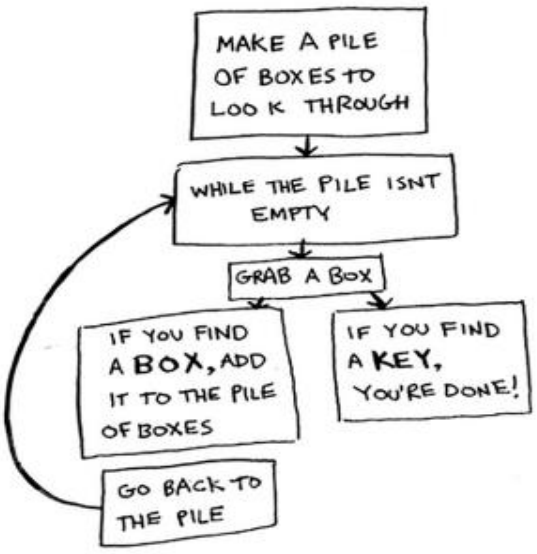


تخبرك جدتك أن مفتاح الحقيبة ربما يكون في هذا الصندوق الآخر.



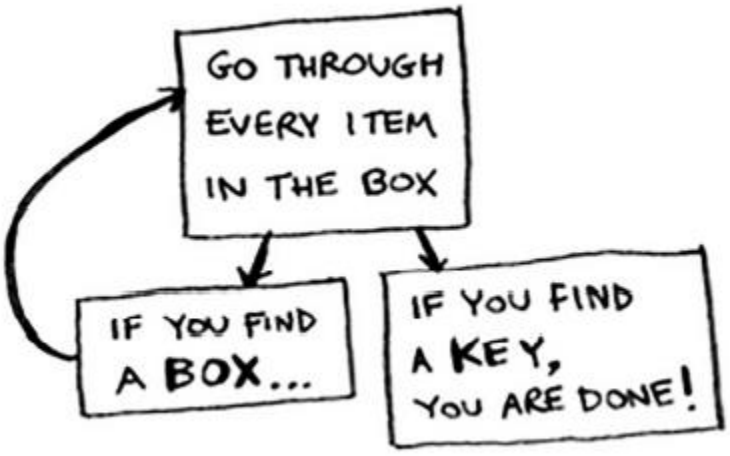
يحتوي هذا الصندوق على المزيد من الصناديق ، مع وجود المزيد من الصناديق داخل تلك الصناديق .
المفتاح في صندوق في مكان ما .ما هي الخوارزمية الخاصة بك للبحث عن المفتاح؟ فكر في خوارزمية قبل أن تستمر في القراءة.

إليك هذه الطريقة :



1. كون مجموعة من الصناديق تبحث من خلالها.
2. احصل على صندوق ، وانظر من خلاله .
3. إذا وجدت صندوقًا ، فأضفه إلى المجموعة للنظر فيه لاحقًا.
4. إذا وجدت المفتاح ، تكون قد انتهيت!
5. كرر هذه الخطوات.

إليك طريقة بديلة:



1. ابحث في الصندوق.
2. إذا وجدت مربعًا ، فانتقل إلى الخطوة 1
3. .إذا وجدت المفتاح ، تكون قد انتهيت!

أي واحدة من الطرق اعلاه تبدو اسهل بالنسبة إليك ؟ الطريقة الأولى مستخدمة (**loop)while** طالما المجموعة ليست فارغة أمسك الصندوق وابحث من خلاله

```
def look_for_key(main_box):
  pile = main_box.make_a_pile_to_look_through()
  while pile is not empty:
    box = pile.grab_a_box()
    for item in box:
      if item.is_a_box():
        pile.append(item)
      elif item.is_a_key():
        print "found the key!"
```

الطريقة الثانية مستخدمة ال (**recursion)** . وهو المكان البتنادي فيه الدالة نفسها

- الطريقة الثانية مكتوبة بال (**pseudocode**) (pseudocode هو وصف عالي المستوى للمشكلة التي تحاول حلها , في رموز .وهو يكتب مثل الكود , لكن من المفترض أن يكون أقرب إلى كلام البشر)

```
def look_for_key(box):
  for item in box:
    if item.is_a_box():
      look_for_key(item)  <-------- Recursion!
    elif item.is_a_key():
      print "found the key!"
```

كلا الطريقتين يحققان نفس الشيء ، لكن الطريقة الثانية أوضح بالنسبة لي . يتم استخدام ال (**recursion)** لجعل الحل أكثر وضوحاً . ليس هناك فائدة في الأداء (**performance**) , في الحقيقة **loops** في بعض الأحيان أفضل في الأداء .

أنا أحب هذا الاقتباس لـ Leigh Caldwell في Stack Overflow

“Loops may achieve a performance gain for your program. Recursion may achieve a performance gain for your programmer. Choose which is more important in your situation!”[1]

"قد تحقق الحلقات مكاسب في أداء برنامجك .قد يحقق التكرار مكاسب في الأداء للمبرمج الخاص بك .اختر أيهما أكثر أهمية في حالتك !" [2]

- العديد من الخوارزميات تستخدم الـ **Recursion** لذلك لايد من فهمه

Base case and recursive case :

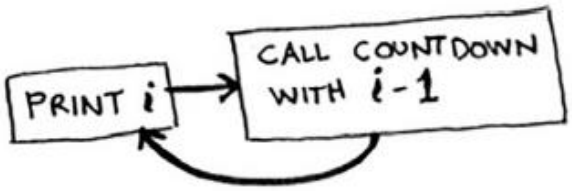
لأن الدالة التكرارية بتستدعي نفسها , من السهل كتابة دالة بشكل غير صحيح تنتهي في حلقة لا نهائية , على سبيل المثال من المفترض أنك تريد أن تكتب دالة تطبع العدد التنازلي كالتالي:

```
> 3...2...1
```

يمكنك كتابتها بشكل متكرر ، هكذا :

```
def countdown(i):
print i
countdown(i-1)
```

إذا كتبت الكود وقمت بتشغيله ستلاحظ المشكلة , ستعمل هذه الدالة إلى ما لا نهاية!



Infinite loop

```
> 3...2...1...0...-1...-2...
```

(اضغط على Ctrl-C لإنهاء النص).

[1] performance - Recursion or Iteration? - Stack Overflow
[2] performance - Recursion or Iteration? - Stack Overflow

عندما تقوم بكتابة دالة تكرارية يجب أن تخبره متى يتوقف عن التكرار. (هذا هو السبب في أن كل دالة تكرارية تتكون من جزأين: الحالة الأساسية والحالة العودية. the base case and the recursive case.)

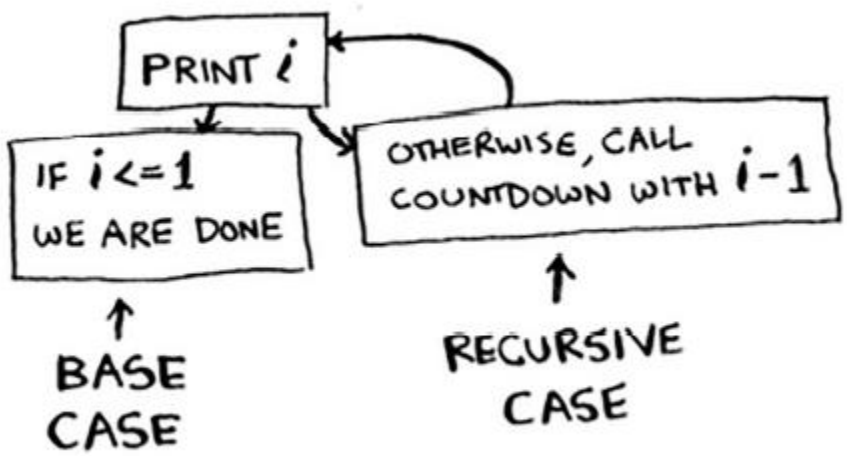
(**The recursive case**) هي عندما تستدعي الدالة نفسها .

(**The base case**) عندما لا تستدعي الدالة نفسها مرة أخرى.

دعنا نضيفها في دالة العد التناولي التي قمنا بكتابتها أعلاه :

```
def countdown(i):
  print i
  if i <= 0:  <------ Base case
    return
  else:  <------ Recursive case
    countdown(i-1)
```

الآن تعمل الدالة كما هو متوقع



☺The stack (المكدسات) :

المكدسات مفهوم مهم في البرمجة , (**The call stack**) مكدس الاستدعاءات هو مفهوم مهم في البرمجة العامة . ومن المهم فهمه عند استخدام الـ recursion .

لنفترض أنك تقيم حفل شواء .تحتفظ بقائمة مهام للشواء ، على شكل مجموعة من الملاحظات اللاصقة.

تذكر مرة أخرى عندما تحدثنا عن المصفوفات والقوائم ، وكان لديك قائمة مهام؟ يمكنك إضافة عناصر المهام في أي مكان إلى القائمة أو حذف العناصر العشوائية .مجموعة الملاحظات اللاصقة أبسط بكثير .عند إدراج عنصر ، تتم إضافته إلى أعلى القائمة .عندما تقرأ عنصرًا ، فأنت تقرأ العنصر الأعلى فقط ، ويتم حذفه من القائمة .لذا فإن قائمة المهام الخاصة بك تحتوي على إجراءين فقط: دفع (إدراج) وانبثاق (إزالة وقراءة). **(push (insert) and pop : (remove and read)**)

PUSH
(ADD A NEW ITEM TO THE TOP)

POP
(REMOVE THE TOPMOST ITEM AND READ IT)

دعنا نرى قائمة المهام قيد التنفيذ

هيكل البيانات هذا يسمى بالمكدس , المكدس عبارة عن هيكل بيانات بسيط.

لقد كنت تستخدم مكدسًا طوال الوقت دون أن تدرك ذلك!

The call stack

يستخدم جهاز الكمبيوتر الخاص بك مكدسًا داخليًا يسمى بـ call stack , دعنا نرى ذلك عمليا , هذه دالة بسيطة

```
def greet(name):
    print "hello, " + name + "!"
    greet2(name)
    print "getting ready to say bye..."
    bye()
```

هذه الدالة تقوم بتحيتك ثم تقوم بإستدعاء دالتين فيما يلي هاتان الدالتين :

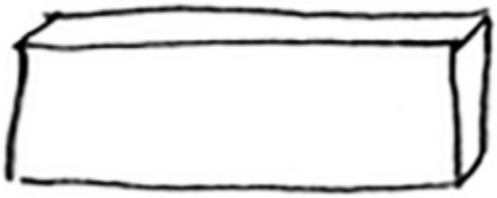
```
def greet2(name):
    print "how are you, " + name + "?"

def bye():
    print "ok bye!"
```

دعنا نتعرف ماذا سيحدث إذا تم إستدعاء إحدى الدوال

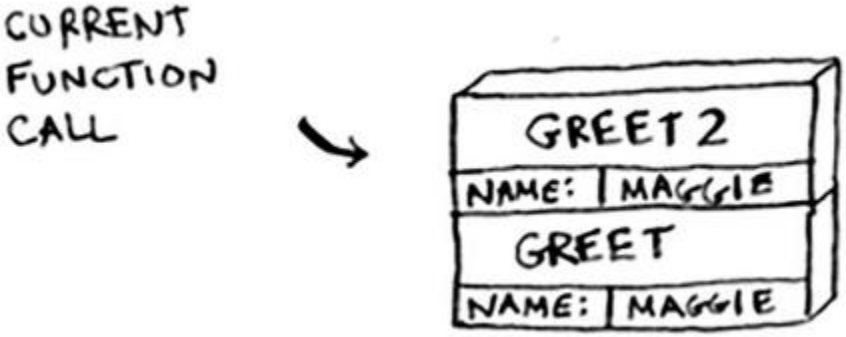
print هي دالة ف بايثون ولكن لتبسيط الأمور في هذا المثال دعنا نتظاهر بأنه ليس كذلك. ☺

فلنفترض أنك استدعيت .(”greet(“maggie, أولا يخصص جهاز الكمبيوتر الخاص بك مكان (مربع) من الذاكرة لاستدعاء هذه الدالة.

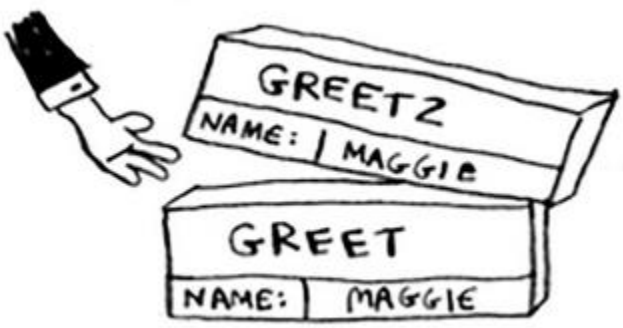


الآن دعونا نستخدم الذاكرة .تم تعيين اسم المتغير على "ماجي" التي يجب حفظها في الذاكرة

في كل مرة تقوم فيها بإستدعاء الدالة , يحفظ جهاز الكمبيوتر الخاص بك القيم لجميع المتغيرات لهذا الاستدعاء في ذاكرة مثل هذه .بعد ذلك ، تقوم بالطباعة !hello, maggie , ثم تستدعي .(”greet2(“maggie , مرة أخرى ، يخصص جهاز الكمبيوتر الخاص بك مربعًا من الذاكرة لاستدعاء هذه الدالة .



يستخدم جهاز الكمبيوتر الخاص بك مكدس لهذه الصناديق , يضاف المربع الثاني فوق الأول , أنت تطبع how are you, maggie? ثم تعود من استدعاء الدالة .عندما يحدث هذا ، يتم إخراج الصندوق الموجود أعلى المكدس.



الآن المربع الأعلى في المكدس هو لدالة greet وهذا يعني أنك عدت إلى دالة الترحيب greet . عندما استدعيت دالة greet2 دالة greet تم الإنتهاء منها جزئيا . وهذه هي الفكرة الكبيرة في هذا القسم (عند استدعاء دالة من دالة أخرى ، يتم إيقاف دالة الاستدعاء مؤقتًا في حالة مكتملة جزئيًا) , لا تزال جميع قيم المتغيرات الخاصة بهذه الدالة مخزنة في الذاكرة. الآن بعد أن انتهيت من دالة greet2 , وعدت إلى دالة greet وتتابع من حيث توقفت .تطبع أولاً getting ready to say bye يمكنك إستدعاء دالة bye

تتم إضافة مربع لهذه الوظيفة إلى الجزء العلوي من المكدس. من ثم تطبع !ok bye والعودة من استدعاء الدالة

وستعود إلى دالة greet لا يوجد شيء آخر يمكن القيام به ، لذلك تعود من دالة greet أيضا . تستخدم هذه المكدسات لحفظ المتغيرات لدوال متعددة , وهذه تسمى بالـ call stack

تمرين :

- افترض انني عرضت عليك call stack مثل هذا

ما هي المعلومات التي يمكن أن تزودني بها ، بناءً على call stack هذا فقط؟

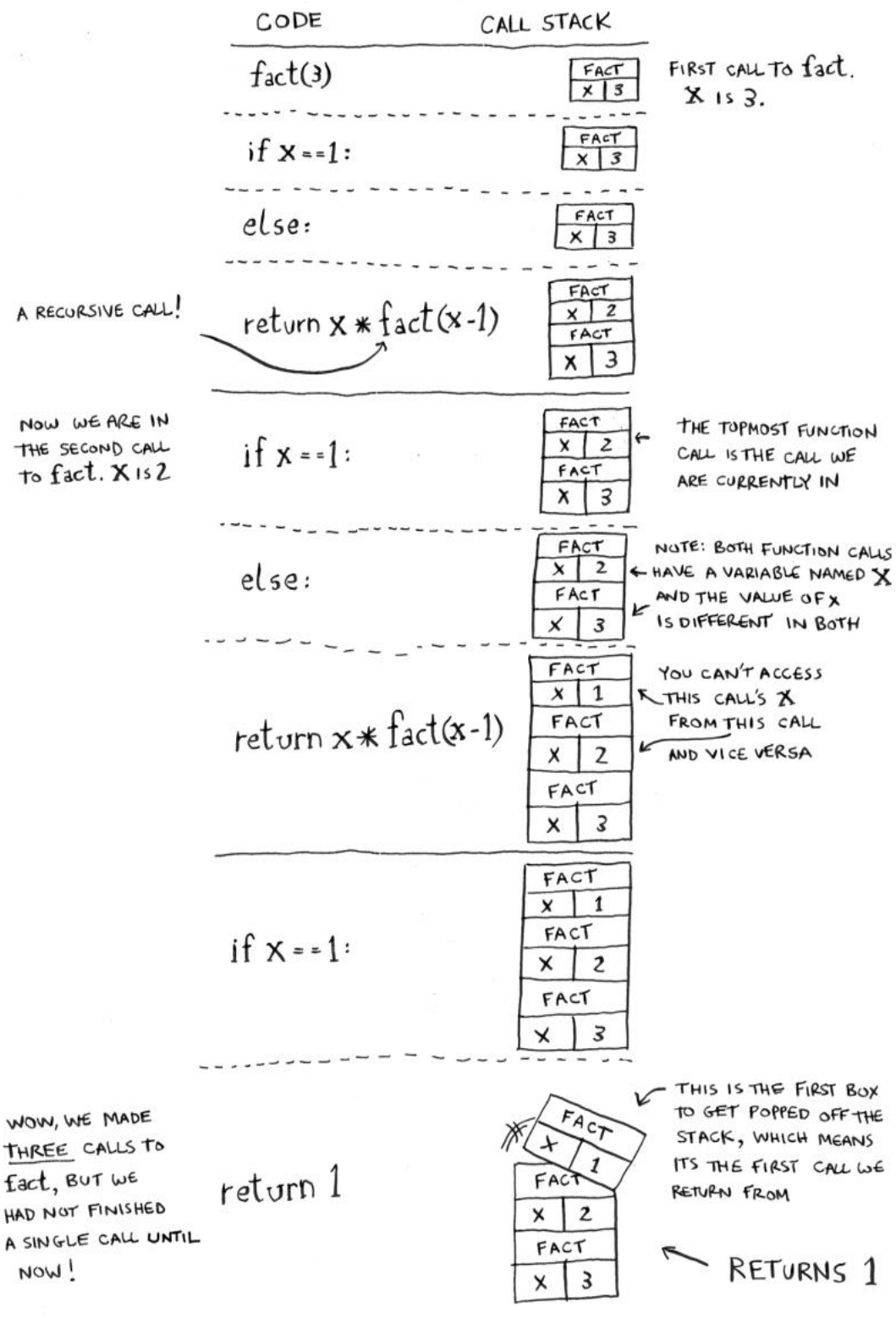
الآن دعونا نرى call stack عمليا مع دالة تكرارية.

The call stack with recursion

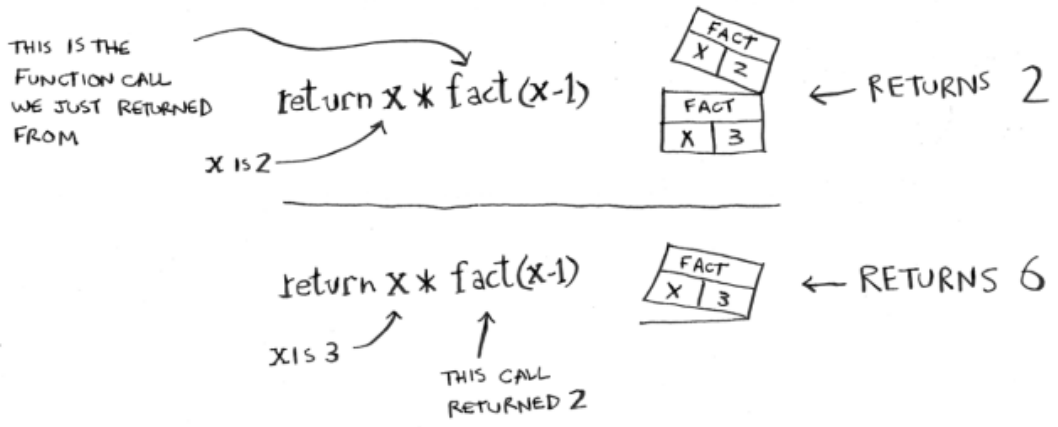
الدوال التكرارية ايضا تستخدم الـ call stack ! دعونا نلقى نظرة على هذا عمليا مع دالة المضروب (factorial) مضروب 5 مكتوبة كـ !5 , ويتم تعريفه على هذا النحو: .1 * 2 * 3 * 4 * 5 = !5 بصورة مماثلة factorial(3) هو 1 * 2 * 3 . فيما يلي دالة تكرارية لحساب مضروب الرقم:

```
def fact(x):
  if x == 1:
    return 1
  else:
    return x * fact(x-1)
```

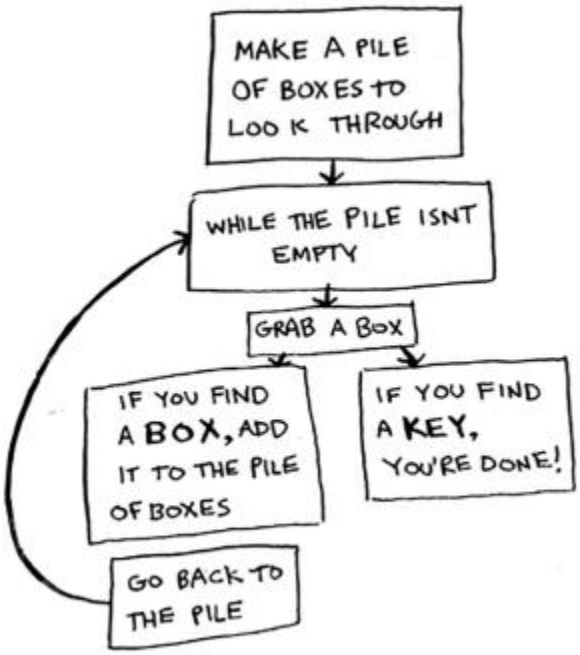
الأن استدعيت .fact(3) دعنا نخطو خلال هذا الاستدعاء سطرًا بسطر ونرى كيف يتغير المكدس. تذكر أن المربع العلوي في المكدس يخبرك باستدعاء الدالة fact التي تعمل عليها حاليًا.


CODE
CALL STACK
fact(3)
FACT
X 3
FIRST CALL TO fact. X IS 3.
if x==1:
else:
A RECURSIVE CALL!
return x * fact(x-1)
FACT
X 2
FACT
X 3
NOW WE ARE IN THE SECOND CALL TO fact. X IS 2
if x==1:
THE TOPMOST FUNCTION CALL IS THE CALL WE ARE CURRENTLY IN
else:
NOTE: BOTH FUNCTION CALLS HAVE A VARIABLE NAMED X AND THE VALUE OF X IS DIFFERENT IN BOTH
return x * fact(x-1)
FACT
X 1
YOU CAN'T ACCESS THIS CALL'S X FROM THIS CALL AND VICE VERSA
if x==1:
WOW, WE MADE THREE CALLS TO fact, BUT WE HAD NOT FINISHED A SINGLE CALL UNTIL NOW!
return 1
THIS IS THE FIRST BOX TO GET POPPED OFF THE STACK, WHICH MEANS ITS THE FIRST CALL WE RETURN FROM
RETURNS 1

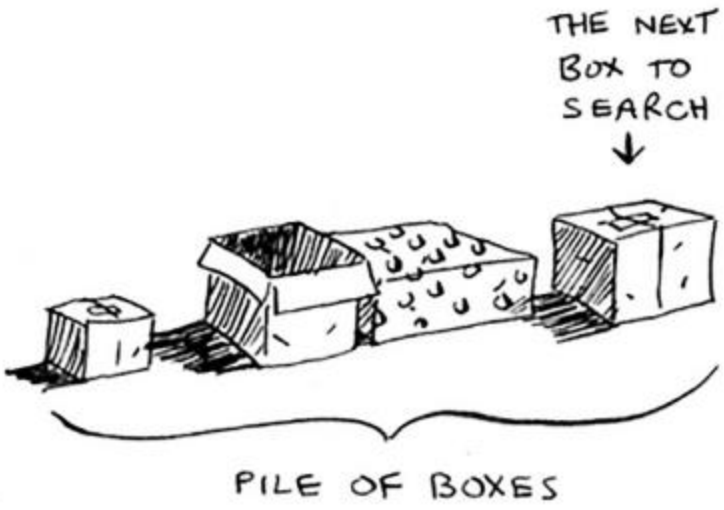
The stack



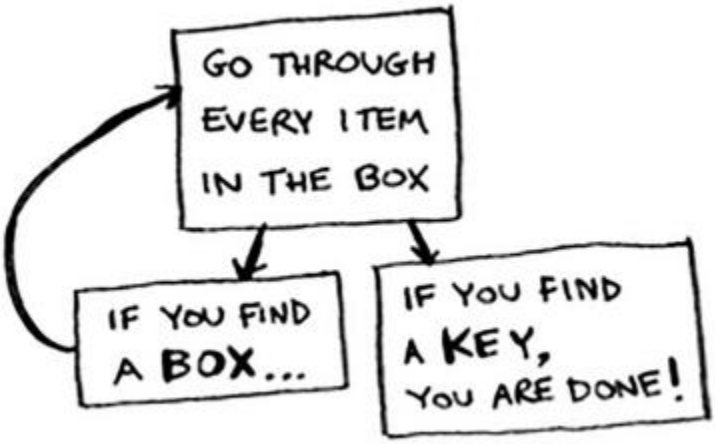
لاحظ أن كل استدعاء لـ fact لديها نسختها الخاصة من x لا يمكنك الوصول إلى نسخة دالة مختلفة من.x .
المكدس يلعب دورًا كبيرًا في العودية .في المثال الافتتاحي ، كان هناك طريقتان للعثور على المفتاح .ها هي الطريقة الأولى مرة أخرى.



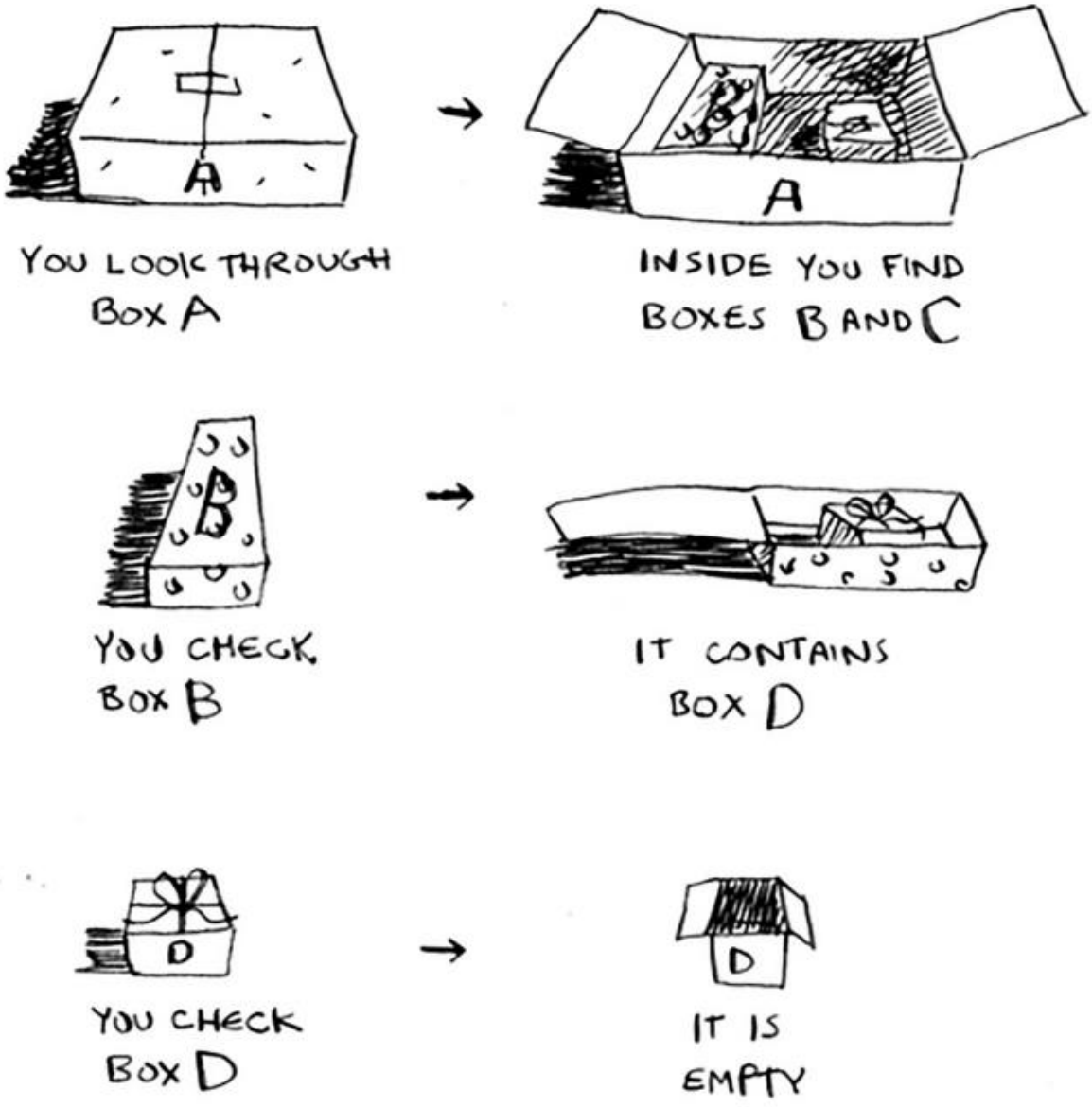
بهذه الطريقة ، يمكنك إنشاء كومة (مجموعة) من المربعات للبحث خلالها ، بحيث تعرف دائمًا المربعات التي لا تزال بحاجة إلى البحث فيها.



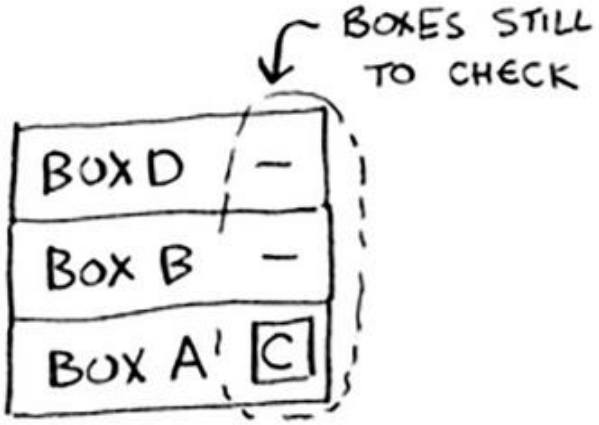
ولكن في النهج التكراري ، لا توجد كومة(مجموعة).



إذا لم يكن هناك كومة (مجموعة) ، كيف تعرف الخوارزمية الخاصة بك المربعات التي لا يزال يتعين عليك البحث فيها؟ هذا مثال



في هذه المرحلة ، يبدو الـ call stack هكذا.



- يتم حفظ "كومة الصناديق" على المكدس! هذه كومة من استدعاءات الوظائف نصف المكتملة ، ولكل منها قائمة نصف كاملة من المربعات للبحث فيها. يعد استخدام المكدس مناسبًا لأنك لست مضطرًا إلى تتبع كومة من الصناديق بنفسك - فالمكدس يقوم بذلك نيابة عنك.
- يعد استخدام المكدس أمرًا مناسبا ، ولكن هناك تكلفة: حفظ كل هذه المعلومات يمكن أن يستهلك قدرًا كبيرًا من الذاكرة. كل استدعاء من هذه الدوال يشغل حيز من الذاكرة وعندما تكون مجموعتك طويلة جدًا ، فهذا يعني أن جهاز الكمبيوتر الخاص بك يقوم بحفظ المعلومات للعديد من استدعاءات الدوال .في هذه المرحلة ، لديك خياران:
  - يمكنك إعادة كتابة التعليمات البرمجية الخاصة بك لاستخدام حلقة بدلا من ذلك.
  - يمكنك استخدام شيء يسمى تكرارية الذيل .هذا موضوع تكراري متقدم خارج نطاق هذا الكتاب .وهي مدعومة أيضًا ببعض اللغات وليس كلها.

تمرين :

افترض أنك كتبت عن طريق الخطأ دالة تكرارية تعمل إلى الأبد .كما رأيت ، يقوم جهاز الكمبيوتر الخاص بك بتخصيص ذاكرة على المكدس لكل استدعاء وظيفي .ماذا يحدث للمكدس عندما تعمل الوظيفة العودية إلى الأبد؟

الخـلاصة :

- Recursion هي عندما تستدعي الدالة نفسها .
- كل دالة تكرارية لها حالتان: الحالة الأساسية والحالة العودية.(the base case : and the recursive case)
- المكدس له عمليتان: دفع وفرقعة. (push and pop)
- تنتقل كل استدعاءات الدوال إلى call stack.
- يمكن أن يصبح call stack كبيرًا جدًا ، مما يستهلك قدرًا كبيرًا من الذاكرة