لديك قائمة من الأرقام [1, 2, 3]، وتريد إيجاد جميع التباديل الممكنة لهذه الأرقام باستخدام **Backtracking**.

الكود:

```
def backtrack(path, options):
    if not options:
        print(path)
        return
    for i in range(len(options)):
        new_path = path + [options[i]]
        new_options = options[:i] + options[i+1:]
        backtrack(new_path, new_options)

backtrack([], [1, 2, 3])
```

شرح الكود

تعريف الدالة backtrack(path, options) :

- **المعامل path:**
  - يمثل الترتيب الحالي الذي يتم بناؤه (مسار التقدم في إنشاء التوافيق).
  - يبدأ كقائمة فارغة [] ويُضاف إليه عنصر جديد في كل خطوة.
- **المعامل options:**
  - يحتوي على العناصر المتبقية التي لم تُستخدم بعد في الترتيب.
  - في كل خطوة، يتم تقليل الخيارات عن طريق إزالة العنصر الذي يُستخدم في هذه الخطوة.

الحالة الأساسية:

```
python
Copy code
if not options:
    print(path)
    return
```

- إذا أصبحت قائمة optionsفارغة، فهذا يعني أنه قد تم بناء ترتيب كامل.
- يتم طباعة الترتيب الحالي (path) ثم التوقف عن المزيد من العمليات لهذه الحالة.

الحالة التكرارية:

```
python
Copy code
for i in range(len(options)):
    new_path = path + [options[i]]
    new_options = options[:i] + options[i+1:]
    backtrack(new_path, new_options)
```

1. يتم تكرار العملية على كل عنصر في قائمة options.
2. لكل عنصر:
   - **new_path**:
     - مسار جديد يتم بناؤه بإضافة العنصر الحالي إلى المسار القديم.
   - **new_options**:
     - قائمة جديدة من الخيارات تُستبعد منها العنصر الحالي.
3. يتم استدعاء الدالة نفسها (backtrack) مع القيم الجديدة.

تنفيذ الكود

عندما يتم استدعاء backtrack([], [1, 2, 3]):

- يتم البدء بمسار فارغ []وقائمة الخيارات .[1, 2, 3]
- عبر التكرار، تُبنى جميع التوافيق الممكنة للأرقام 1، 2، و3.

النتيجة:

الكود يولد جميع الترتيبات الممكنة:

csharp
Copy code
[1, 2, 3]
[1, 3, 2]
[2, 1, 3]
[2, 3, 1]
[3, 1, 2]
[3, 2, 1]

الفكرة الأساسية:

- في كل خطوة:
  1. نضيف عنصرًا جديدًا إلى المسار.(path)
  2. نزيل هذا العنصر من الخيارات المتبقية.(options)
  3. نكرر العملية حتى تكون الخيارات فارغة.(Base Case)
- بهذه الطريقة، يتم استكشاف جميع التفرعات الممكنة لتوليد الترتيبات.

مثال: إيجاد التراكيب (Combinations) التي يكون مجموعها مساويًا لرقم معين

```
def find_combinations(target, path, options, start):
# إذا كان المجموع الحالي يساوي الهدف
    if sum(path) == target:
        print(path)
        return
# إذا كان المجموع الحالي أكبر من الهدف، توقف
    if sum(path) > target:
        return
# التكرار عبر الخيارات
    for i in range(start, len(options)):
# إضافة العنصر الحالي إلى المسار
        new_path = path + [options[i]]
# استدعاء الباك تراكينغ مع المسار الجديد
        find_combinations(target, new_path, options, i)


# اختبار الدالة
numbers[3 ,2 ,1] =
target_sum = 4
find_combinations(target_sum, [], numbers, 0)
```

شرح الكود:

1. **المعاملات:**
   - target: الهدف (المجموع المطلوب).
   - path: يمثل التركيب الحالي الذي يتم بناؤه.
   - options: قائمة الأرقام المتاحة.
   - start: المؤشر الحالي لضمان عدم إعادة استخدام الأرقام السابقة.
2. **الحالة الأساسية:**
   - إذا كان المجموع الحالي (sum(path)) يساوي الهدف، تتم طباعة path.
3. **الحالة التوقفية:**
   - إذا كان المجموع الحالي أكبر من الهدف، يتم التوقف لأن المسار لم يعد صالحًا.
4. **التكرار:**
   - يبدأ التكرار من الموضع الحالي (start) لضمان عدم إعادة استخدام الأرقام السابقة.
   - يتم استدعاء الدالة نفسها مع:
     - المسار الجديد (new_path).
     - نفس الخيارات.
     - مؤشر البداية المحدث (i).

النتائج عند التنفيذ:

عند استدعاء: find_combinations(4, [], [1, 2, 3], 0)

- يبدأ الكود بمحاولة جميع التراكيب التي يكون مجموعها مساويًا لـ 4.

النتائج:

csharp

Copy code

[1, 1, 1, 1]

[1, 1, 2]

[1, 3]

[2, 2]

الفكرة الأساسية:

- يتم استخدام **الباك تراكينغ** هنا لتجربة جميع التراكيب الممكنة للأرقام الموجودة في القائمة.
- في كل خطوة، إذا تجاوز المجموع الهدف، يتم التوقف عن الاستمرار في هذا المسار.
- بهذه الطريقة، نضمن توليد جميع التراكيب الصالحة فقط.