Создаем новый тип (этакий рекурсивный вариант)

```
type 'a new_list =
        | Item of 'a
        | List of 'a new_list list
;;
```

Пояснение:

- type 'a new_list - строка означает, что new_list - это тип с параметром 'a (может быть и несколько таких параметров если нужно). Мы параметризировали тип)
- Item of 'a - здесь Item - это конструктор, а “'a” - это тип его аргумента. Данный конструктор представляет один простой элемент списка (число).
- List of 'a new_list list - здесь List конструктор, а его аргумент будет с типом “'a new_list list”. Эта непонятная штука означает, что аргумент будет типа “list”, который будет хранить элементы типа “'a new_list” (а они могут быть представлены как Item или опять же List). Таким образом я создал “рекурсивный вариант”, что позволяет в списке хранить либо списки, либо одиночные элементы.

Варианты обычно используют для создания деревьев. Подробно о вариантах стоит (“must” я бы сказал) почитать:

1) https://habrahabr.ru/post/108920/ - с раздела варианты и лучше до конца;
2) В книге “Chailloux Manoury Pagano - Developing Applications With Objective Caml” написано доступно: на страницах 45-48 от **Sum types** до **Parametrized types** (но я бы рекомендовал дочитать до 52 страницы чтобы охватить инфу о деревьях).

Пример создания списка такого типа

```
List[ List [Item(1);Item(2);]; Item(1)];; (*аналогично: [[1;2];[1]]*)
```

Для примера работы, я решил реализовать суммирование элементов списка, учитывая подсписки и обычные элементы:

```
let rec sum newl =
        match newl with
        | Item(x)  -> x
        | List([]) -> 0
        | List(head::tail) -> (sum head) + (sum (List(tail)))
;;
```

В объяснении я опишу только (sum (List(tail))). Я сначала использовал просто tail, но окамль ругался. Есть просто такая ситуация: “Item(2); Item(3);”. Что они из себя представляют - непонятно( Чтобы объяснить окамлю, что это именно список, нужно указать List(tail). Да, и со всеми скобочками. Иначе окамль будет говорить, что у тебя слишком много параметров задано.

Примеры для суммы

```
sum (List[ List [Item(1);Item(2);]; Item(1)]);;
sum (List[ List [Item(1);List[Item(2);Item(1)];]; Item(1)]);;
sum (List[]);;
sum (Item(4));; (*сумма одного простого элемента - но зачем? :)*)
```