Data Abstraction

Announcements

Discussion 4

Max Product

Write a function that takes in a list and returns the maximum product that can be formed using non-consecutive elements of the list. All numbers in the input list are greater than or equal to 1.

```
def max_product(s):
    """Return the maximum product that can be
    formed using non-consecutive elements of s.

    >>> max_product([10, 3, 1, 9, 2]) # 10 * 9
    90
    >>> max_product([5, 10, 5, 10, 5]) # 5 * 5 * 5
    125
    >>> max_product([])
    1
    """
    if len(s) == 0:
        return 1
    elif len(s) == 1:
        return s[0]
    else:
        return ____
```

A tip for finding a recursive process:

1. Pick an example: s = [5, 10, 5, 10, 5]
2. Write down what recursive calls will do:
   - max_product([10, 5, 10, 5]) → 10 * 10
   - max_product([5, 10, 5]) → 5 * 5
   - max_product([10, 5]) → 10
   - max_product([5]) → 5
3. Which one helps build the result?

Either include s[0] but not s[1], OR
Don't include s[0]

Choose the larger of:
multiplying s[0] by the max_product of s[2:] (skipping s[1]) OR
just the max_product of s[1:] (skipping s[0])

```
max(s[0] * max_product(s[2:]), max_product(s[1:]))
```

Sum More Fun

Implement `nested_sums(n)`, which takes a total `n>0`. It returns a list of all nested lists of `n` 1's that have at least one 1 between each pair of brackets.

Allowed: `[1, [1, 1], 1]`
Not allowed: `[[1, 1, 1], 1]`

No 1 in between these brackets!

```
def nested_sums(n):
    """Return all nested lists of n 1's with no adjacent brackets.

    >>> for s in nested_sums(5): print(s)
    [1, 1, 1, 1, 1]
    [1, 1, 1, [1], 1]
    [1, 1, [1], 1, 1]
    [1, 1, [1, 1], 1]
    [1, [1], 1, 1, 1]
    [1, [1], 1, [1], 1]
    [1, [1, 1], 1, 1]
    [1, [1, 1, 1], 1]
    [1, [1, [1], 1], 1]
    """
    if n < 0:
        return []
    if n == 0:
        return [[]]
    result = [[1] + rest for rest in nested_sums(n-1)]
             # E.g., make [1, 1, 1] from [1, 1]
    for k in range(1, n-1):
        for nest in nested_sums(k):
            result = result + ___
    return result
```

For n=5, nested_sums(n-1) has:
[1, 1, 1, 1]
[1, 1, [1], 1]
[1, [1], 1, 1]
[1, [1, 1], 1]

Build all the nested sums of the form [1, [...], ...] where the inner list has k 1's.

Max and Min

Key Function for Max and Min

```
>>> s = [-3, -5, -4, -1, -2]
>>> max(s)
-1
>>> max(s, key=abs)
-5
>>> max([abs(x) for x in s])
5
```

Example: Two Lists

Given these two related lists of the same length:

```
xs = range(-10, 11)
ys = [x*x - 2*x + 1 for x in xs]
```

Write an expression that evaluates to the x for which the corresponding y is smallest:

```
>>> list(xs)
[-10,  -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4,  5,  6,  7,  8,  9, 10]
>>> ys
[121, 100, 81, 64, 49, 36, 25, 16,  9,  4, 1, 0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> x_corresponding_to_min_y
1
```