An interesting problem...

Write a Python function called `xover` that takes the following parameters: one or two lists of bits, and a probability `indpb` (as a real number in [0.0, 1.0]). Let the first list be called `p1` and the second `p2`. `p2` and `indpb` are optional. Their default values are `None` and 0.5, respectively. It does not return a value.

`xover` goes through `p1`, replacing each bit with the corresponding bit from `p2` with probability `indpb`, if `p2` was provided. Otherwise, bits in `p1` are replaced with 1 with probability `indpb`. 
Solution 1

```python
def xover(p1, p2=None, indpb=0.5):
    if p2 is None:
        p2 = [1] * len(p1)

    for i in range(len(p1)):
        if random() < indpb:
            p1[i] = p2[i]
```

Solution-ish 2 – using an iterator

```python
def xover(p1, p2=None, indpb=0.5):
    if p2 is None:
        p2 = [1] * len(p1)

    for e in p1:
        if random() < indpb:
            e = <something>

    Hmm, what goes here?
```
Solution-ish 3 – using an iterator

Problem: confusion about the difference between an index and a value:

```python
def xover(p1, p2=None, indpb=0.5):
    if p2 is None:
        p2 = [1] * len(p1)

    for e in p1:
        if random() < indpb:
            p1[e] = p2[e]

    # e is a value in p1 NOT an index into p1!
```

Solution-ish 4 – comprehension anyone?

Could we use a comprehension? Wouldn’t it be complicated?

Sort of and yes.

```python
p1 = [p1[i] if random() < indpb else 1 if p2 is None else p2[i] for i in range(len(p1))]
```

This works! Almost.
But it’s a little complicated.
Let’s break this down.

First: there is no filtering in this comprehension!

\[
p_1 = [p_1[i] \text{ if } \text{random()} < \text{indpb} \text{ else } 1 \text{ if } p_2 \text{ is None else } p_2[i] \text{ for } i \text{ in } \text{range}(\text{len}(p_1))]
\]

NOT filtering.

An Aside

Quick review of the format of a comprehension:

\[
p_1 = [\text{put } f(e) \text{ in new list for each } e \text{ in old list if filter}]
\]

filter goes here and **may affect number of elements** placed in the resulting list
Unwinding Solution-ish 4

\[
p1 = [\begin{array}{c}
p1[i] \text{ if } \text{random()} < \text{indpb} \\
\text{else } 1 \text{ if } p2 \text{ is None} \\
\text{else } p2[i]
\end{array}
\text{ for } i \in \text{range}(\text{len}(p1))]\
\]

What goes in new list

Iteration of old list

Unwinding Solution-ish 4

\[
p1[i] \text{ if } \text{random()} < \text{indpb} \quad \text{else} \quad 1 \text{ if } p2 \text{ is None} \quad \text{else} \quad p2[i]
\]

Mutually exclusive conditional logic to determine what is put in the new list

Logically equivalent but not valid syntax

\[
\begin{align*}
\text{if } \text{random()} < \text{indpb} & \quad \text{add } p1[i] \text{ to list} \\
\text{else if } p2 \text{ is None} & \quad \text{add 1 to list} \\
\text{else} & \quad \text{add } p2[i] \text{ to list}
\end{align*}
\]

determines what is added

NOT if something is added
A Simpler Example

Record coin flips

flips = ['T' if random() < 0.5 else 'H' for _ in range(20)]

Another Simpler Example

Another example: take even numbered elements from L1 and odd numbered elements from L2, where L1 and L2 have the same length

new = [L1[i] if i % 2 == 0 else L2[i] for i in range(len(L1))]}
Unwinding Solution-ish 4

Returning to our question:
Could we use a comprehension? Wouldn’t it be complicated?

\[ p1 = \begin{cases} p1[i] & \text{if } \text{random()} < \text{indpb} \text{ else } 1 \text{ if } p2 \text{ is None} \\ & \text{else } p2[i] \text{ for } i \text{ in range}(\text{len}(p1)) \end{cases} \]

Why does this only “sort of” work?

The comprehension works. But to use it in our function, we have to reassign \( p1 \), thus we are no longer changing the list in the calling context.

Maps

In Python, a map provides another way to apply a function to each element of an iterable (list, tuple, etc.)

```python
def convert(deg_c):
    return deg_c * 1.8 + 32

f_list = map(convert, c_list)
```

apply this function to this list
Map Example 2

Let u_lists be a list of unsorted lists of integers. Use map to create a list containing the same sublists but with their elements in sorted order:

\[
s\_lists = \text{map}(\text{sorted}, \ u\_lists)
\]
Map Example 4

Using an ad hoc function applied to a list of integers:

\[
\text{polys} = \text{map}(\lambda x: 2x + 4, \text{int\_list})
\]

Zip

Create a list of tuples from some number of other lists:

\[
\text{result\_tuples} = \text{zip}(\text{list1, list2, ..., listn})
\]

- result\_tuples is a list
- each element is a tuple
- each tuple contains n elements – one from each list
Zip Example 1

```python
digits = [1, 2, 3]
words = ['one', 'two', 'three']
romans = ['i', 'ii', 'iii']

combos = zip(digits, words, romans)

combos: [(1, 'one', 'i'), (2, 'two', 'ii'), (3, 'three', 'iii')]
```

Zip Example 2

Take an unsorted list and create a list of tuples that contain the values and their position in the original list:

```python
nums = unsorted list of n integers

order = zip(nums, [i for i in range(len(nums))])
sorted_order = sorted(order, key=lambda x: x[0])
```