Quiz 3 – Problem 2

Write a Python function that takes the following parameters: one or two lists of bits, and a probability \( \text{indpb} \) (as a real number in \([0.0, 1.0]\)). Call the first list \( p1 \) and the second \( p2 \). \( p2 \) and \( \text{indpb} \) are optional. Their default values are None and 0.5, respectively. It then goes through \( p1 \), replacing each bit with the corresponding bit from \( p2 \) with probability \( \text{indpb} \), if \( p2 \) was provided. Otherwise, bits in \( p1 \) are replaced with 1 with probability \( \text{indpb} \).

```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]
```

Some of you did something like this:

```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>
```

This reassigns \( e \) but does not change the value in the list!
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]

There is still some confusion about the difference between an index and a value:

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

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

Sort of and yes.

This works! Sort of.

Quick review of the format of a comprehension:

- `filter goes here and may affect number of elements placed in the resulting list`
Let's break this down.

First: there is no filtering in this comprehension!

\[
p_1 = [p_1[i] \text{ if } \text{random()} < \text{indpb} \text{ else } 1 \text{ if } \text{p}_2 \text{ is None} \text{ else } \text{p}_2[i]]
\]

for \(i\) in range(len(p1))

What goes in new list

Iteration of old list

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

\[ \text{if random()} < \text{indpb} \text{ add } p_1[i] \text{ to list} \]
\[ \text{else if } \text{p}_2 \text{ is None} \text{ add 1 to list} \]
\[ \text{else add } p_2[i] \text{ to list} \]

logically equivalent but not valid syntax

An example: record coin flips

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

Another example: take even numbered elements from \(L_1\) and odd numbered elements from \(L_2\), where \(L_1\) and \(L_2\) have the same length

\[
\text{new} = [L_1[i] \text{ if } i % 2 == 0 \text{ else } L_2[i] \text{ for } i \text{ in range(len(L_1))}]
\]
Quiz 3 – Problem 2 – An exploration

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

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

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.).

\[
\text{def convert(deg}_c): \\
\quad \text{return } deg_c \times 1.8 + 32 \\
\]

\[ f\text{\_list} = \text{map(convert, c\_list)} \]

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:

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

Map Example 3

Find distances between corresponding locations in a list of starting points and a list of destinations:

\[
\text{def distance}(pt1, pt2): \\
\quad \text{dx} = pt1[0] - pt2[0] \\
\quad \text{dy} = pt1[1] - pt2[1] \\
\quad \text{return } \text{math.sqrt(dx**2 + dy**2)}
\]

\[ \text{dists} = \text{map(distance, sources, dests)} \]
Map Example 4

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

```python
polys = map(lambda x: 2*x + 4, int_list)
```

Zip

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

```python
result_tuples = zip(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])
```