Q3: (Tutorial) Inheritance Review: That's a Constructor, __init__? (1/2)

Let's say we want to create a class `Monarch` that inherits from another class, `Butterfly`. We've partially written an `__init__` method for `Monarch`. For each of the following options, state whether it would correctly complete the method so that every instance of `Monarch` has all of the instance attributes of a `Butterfly` instance? You may assume that a monarch butterfly has the default value of 2 wings.

```
class Butterfly():
    def __init__(self, wings=2):
        self.wings = wings

class Monarch(Butterfly):
    def __init__(self):
        _____________________________________________
        self.colors = ['orange', 'black', 'white']
```

1. `super.__init__()`

2. `super().__init__()`

3. `Butterfly.__init__()`

4. `Butterfly.__init__(self)`

Q3: (Tutorial) Inheritance Review: That's a Constructor, __init__? (2/2)

Some butterflies like the Owl Butterfly have adaptations that allow them to mimic other animals with their wing patterns. Let's write a class for these `MimicButterflies`. In addition to all of the instance variables of a regular `Butterfly` instance, these should also have an instance variable `mimic_animal` describing the name of the animal they mimic. Fill in the blanks in the lines below to create this class.

```
class Butterfly():
    def __init__(self, wings=2):
        self.wings = wings

class Monarch(Butterfly):
    def __init__(self):
        super().__init__()
        self.colors = ['orange', 'black', 'white']



# FILL THIS IN
class MimicButterfly(____________________):

    def __init__(self, mimic_animal):

        ____________________________.init()

        ____________________________ = mimic_animal
```

Q4: (Tutorial) Warmup: The Hy-rules of Linked Lists

In this question, we are given the following Linked List:

```
ganondorf = Link('zelda', Link('young link', Link('sheik', Link.empty)))
```

What expression would give us the value 'sheik' from this Linked List?

What is the value of `ganondorf.rest.first`?

Q5: (Tutorial) Multiply Lnks

Write a function that takes in a Python list of linked lists and multiplies them element-wise. It should return a new linked list.

If not all of the `Link` objects are of equal length, return a linked list whose length is that of the shortest linked list given. You may assume the `Link` objects are shallow linked lists, and that `lst_of_lnks` contains at least one linked list.

```
def multiply_lnks(lst_of_lnks):
    """
    >>> a = Link(2, Link(3, Link(5)))
    >>> b = Link(6, Link(4, Link(2)))
    >>> c = Link(4, Link(1, Link(0, Link(2))))
    >>> p = multiply_lnks([a, b, c])
    >>> p.first
    48
    >>> p.rest.first
    12
    >>> p.rest.rest.rest is Link.empty
    True
    """
    # Implementation Note: you might not need all lines in this skeleton code
    _____________________ = __________

    for ___________________________________________:

        if ___________________________________________:

            ____________________________________

        ___________________

    ____________________________________________________________

    ____________________________________________________________

    # For an extra challenge, try writing out an iterative approach as well!
```

Q6: (Tutorial) Flip Two

Write a recursive function `flip_two` that takes as input a linked list `s` and mutates `s` so that every pair is flipped.

```
def flip_two(s):
    """
    >>> one_lnk = Link(1)
    >>> flip_two(one_lnk)
    >>> one_lnk
    Link(1)
    >>> lnk = Link(1, Link(2, Link(3, Link(4, Link(5)))))
    >>> flip_two(lnk)
    >>> lnk
    Link(2, Link(1, Link(4, Link(3, Link(5)))))
    """
    "*** YOUR CODE HERE ***"



    # For an extra challenge, try writing out an iterative approach as well!
```

Q8: (Tutorial) Find Paths

Hint: This question is similar to `find_paths` on Discussion 05.
Define the procedure `find_paths` that, given a Tree `t` and an `entry`, returns a list of lists containing the nodes along each path from the root of `t` to `entry`. You may return the paths in any order.

For instance, for the following tree `tree_ex`, `find_paths` should behave as specified in the function doctests.

```
def find_paths(t, entry):
    """
    >>> tree_ex = Tree(2, [Tree(7, [Tree(3), Tree(6, [Tree(5), Tree(11)])]), Tree(1, [Tree(5)])])
    >>> find_paths(tree_ex, 5)
    [[2, 7, 6, 5], [2, 1, 5]]
    >>> find_paths(tree_ex, 12)
    []
    """

    paths = []

    if ____________________________________:

        ____________________________________

    for ____________________________________:

        ____________________________:

            ____________________________________________

    ____________________
```