Q2: (Tutorial) Make Keeper

Write a function similar to `keep_ints` like in Question 1 #, but now it takes in a number `n` and returns a function that has one parameter `cond`. **The returned function prints out numbers from 1 to `n` where calling `cond` on that number returns `True`.**

```
def make_keeper(n):
    """
        >>> def is_even(x):
        ...     # Even numbers have remainder 0 when divided by 2.
        ...     return x % 2 == 0
        >>> make_keeper(5)(is_even)
        2
        4
    """
    "*** YOUR CODE HERE ***"
```

Q5: (Tutorial) HOF Diagram Practice

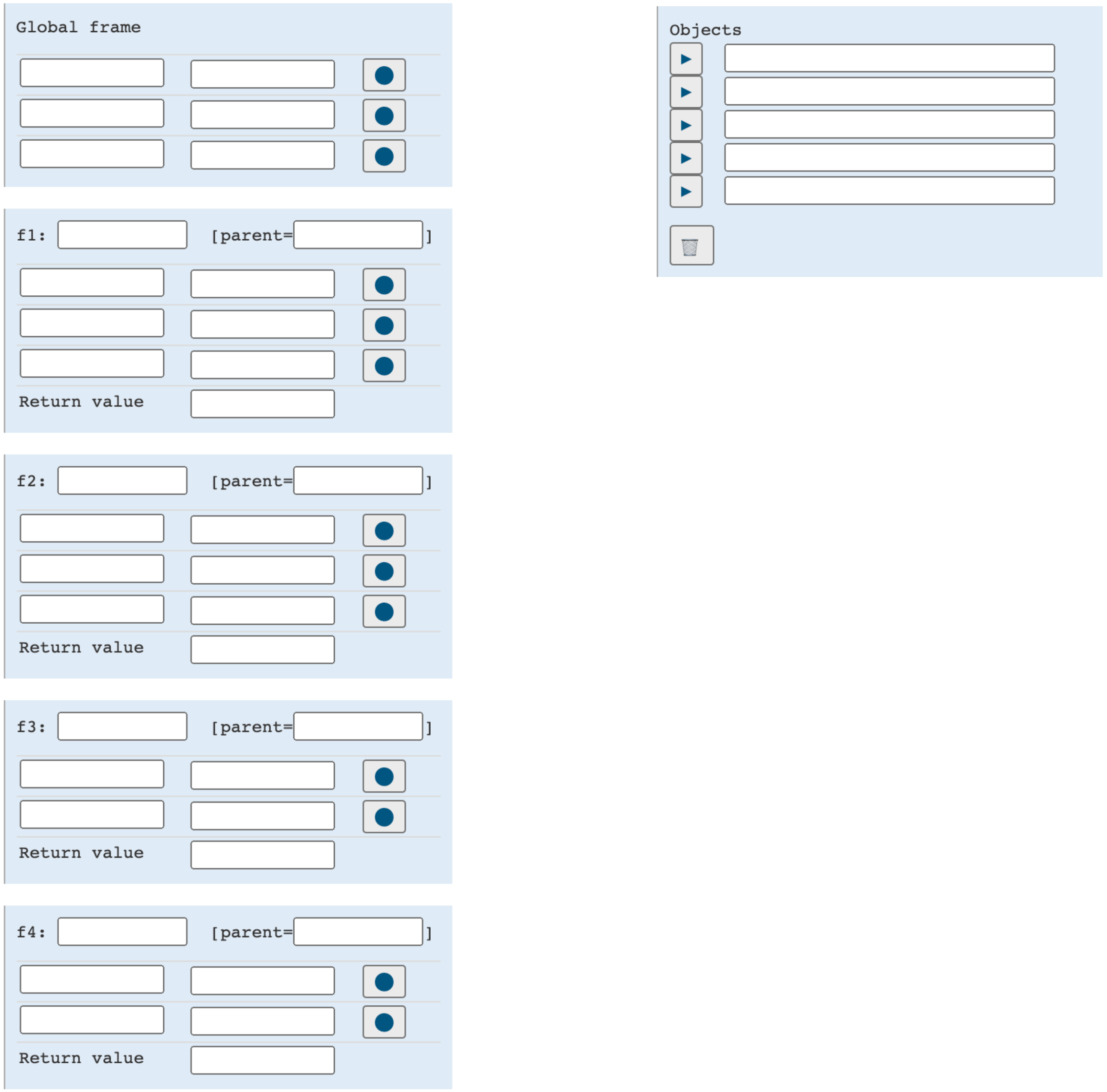
Draw the environment diagram that results from executing the code below

```
n = 7
def f(x):
    n = 8
    return x + 1

def g(x):
    n = 9
    def h():
        return x + 1
    return h

def f(f, x):
    return f(x + n)

f = f(g, n)
g = (lambda y: y())(f)
```



Q7: (Tutorial) Warm Up: Make Keeper Redux

In this question, we will explore the execution of a self-reference function, `make_keeper_redux`, based off Question 2, `make_keeper`. The function `make_keeper_redux` is similar to `make_keeper`, but now the returned function also returns **another function** with the same behavior. Feel free to paste and modify your code for `make_keeper` below.

```
def make_keeper_redux(n):
    """
        >>> def multiple_of_4(x):
        ...     return x % 4 == 0
        >>> def ends_with_1(x):
        ...     return x % 10 == 1
        >>> k = make_keeper_redux(11)(multiple_of_4)
        4
        8
        >>> k = k(ends_with_1)
        1
        11
        >>> k
        <function do_keep>
    """
    # Paste your code for make_keeper here!
```

(Hint: you only need to add one line to your `make_keeper` solution. What is currently missing from `make_keeper_redux`?)

Q9: (Tutorial) Print N

Write a function `print_n` that can take in an integer `n` and returns a repeatable print function that can print the next `n` parameters. After the nth parameter, it just prints "done".

```
def print_n(n):
    """
    >>> f = print_n(2)
    >>> f = f("hi")
    hi
    >>> f = f("hello")
    hello
    >>> f = f("bye")
    done
    >>> g = print_n(1)
    >>> g("first")("second")("third")
    first
    done
    done
    <function inner_print>
    """

    def inner_print(x):

        if ________________________

            print("done")
        else:
            print(x)

        return ____________________

    return ________________________
```