CS61A Exam Prep 2

We will start at Berkeley time!

slides are at go.cs61a.org/melanie-slides

fall 2021 Q3a

Implement `hail_min`, which takes a positive integer `n` and a one-argument function `measure`. It returns the element of the hailstone sequence starting with `n` for which calling `measure` on the element returns the smallest value.

If more than one element of the sequence has the smallest `measure` value, return the earliest one.

```
def hail_min(n, measure):
    """Return the element k of the hailstone sequence starting with n for which
    measure(k) is smallest. In case of a tie, return the earliest element.

    >>> hail_min(5, lambda k: -k)          # Among 5, 16, 8, 4, 2, 1; 16 is largest
    16
    >>> hail_min(8, lambda k: -k)          # Among 8, 4, 2, 1; 8 is largest
    8
    >>> hail_min(3, lambda k: abs(k - 7))  # Among 3, 10, 5, 16, 8, 4, 2, 1; 8 is closest to 7
    8
    >>> hail_min(9, lambda k: abs(k - 7))  # Among 9, 28, 14, 7, 22, ...; 7 is closest to 7
    7
    >>> hail_min(8, lambda k: abs(k - 3))  # 4 and 2 are both close to 3, but 4 is earliest
    4
    """
    apple = _________
              (a)
    while n > 1:
        n = next_hail(n)
        if _________:
             (b)
            _________
               (c)
    return _________
             (d)
```

apple = n

measure(n) < measure(apple)

apple = n

Spring 2021

Q4a

Implement `restrict_domain`, a function that accepts three parameters (`f`, `low_d`, `high_d`) and returns a higher-order function that returns the same thing as `f` when given an argument between `low_d` and `high_d`, inclusive, and otherwise returns `float("-inf")`.

```
def restrict_domain(f, low_d, high_d):
    """Returns a function that restricts the domain of F,
    a function that takes a single argument x.
    If x is not between LOW_D and HIGH_D (inclusive),
    it returns -Infinity, but otherwise returns F(x).

    >>> from math import sqrt
    >>> f = restrict_domain(sqrt, 1, 100)
    >>> f(25)
    5.0
    >>> f(-25)
    -inf
    >>> f(125)
    -inf
    >>> f(1)
    1.0
    >>> f(100)
    10.0
    """
    def wrapper_method_name(n):
        # (a)
        if n >= low_d and n <= high_d:
            # (b)
            return f(n)
        # (c)
        return float("-inf")
    # (d)
    return wrapper_method_name
```

Spring 2021 Q4b

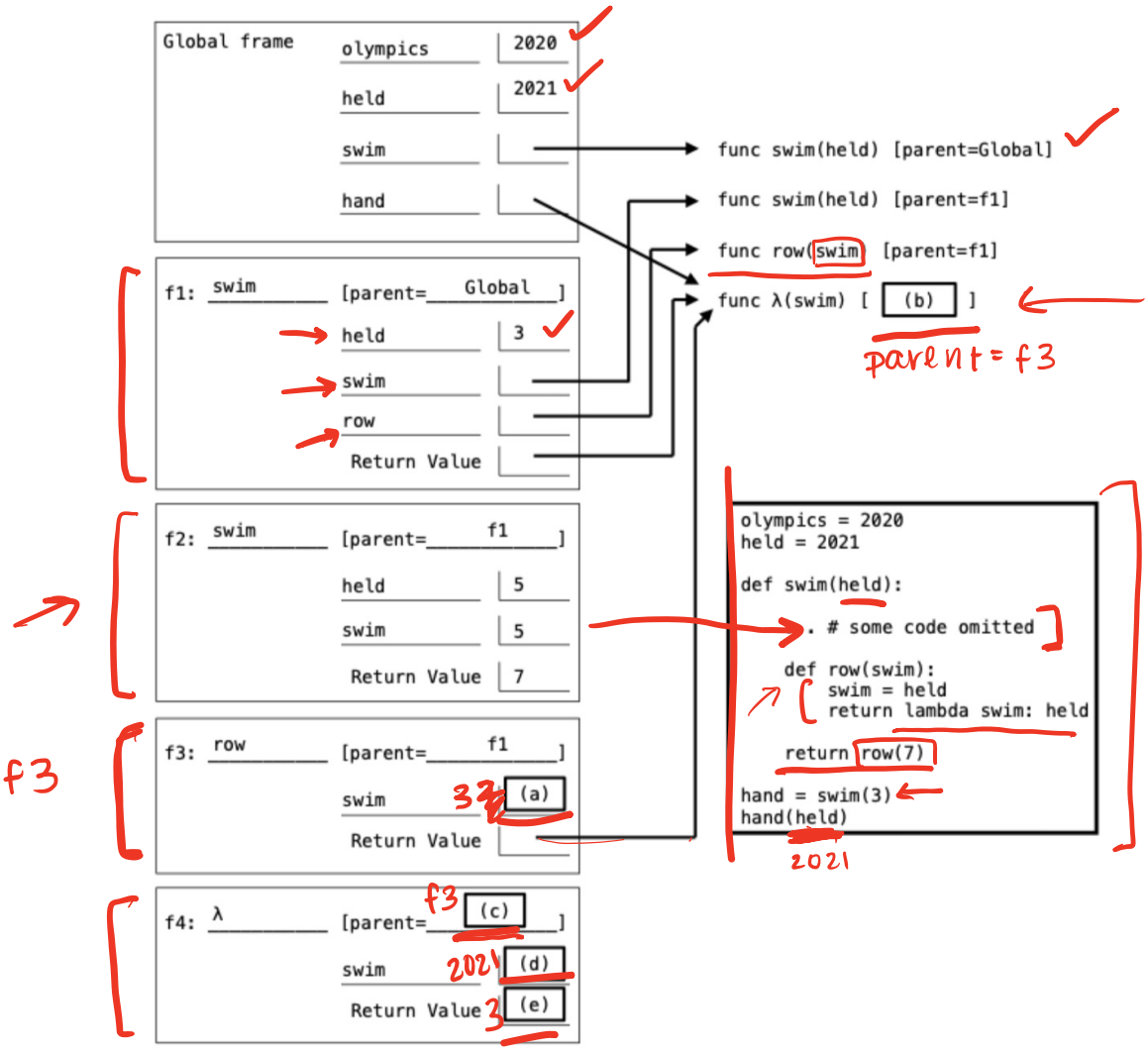
Implement `restrict_range`, a function that accepts three parameters (`f`, `low_r`, `high_r`) and returns a higher-order function that returns the same thing as `f` when that result is between `low_r` and `high_r` (inclusive), and otherwise returns `float("-inf")`.

```
def restrict_range(f, low_r, high_r):
    """Returns a function that restricts the range of F, a function
    that takes a single argument X. If the return value of F(X)
    is not between LOW_R and HIGH_R (inclusive), it returns -Infinity,
    but otherwise returns F(X).

    >>> cube = lambda x: x * x * x
    >>> f = restrict_range(cube, 1, 1000)
    >>> f(1)
    1
    >>> f(-5)
    -inf
    >>> f(5)
    125
    >>> f(10)
    1000
    >>> f(11)
    -inf
    """
    def wrapper_method_name(x):
        #   (a)
        result = f(x)
        #   (b)
        if result < low_r or result > high_r:
            #   (c)
            return float("-inf")
        #   (d)
        return result
        #  (e)
    return wrapper_method_name
```

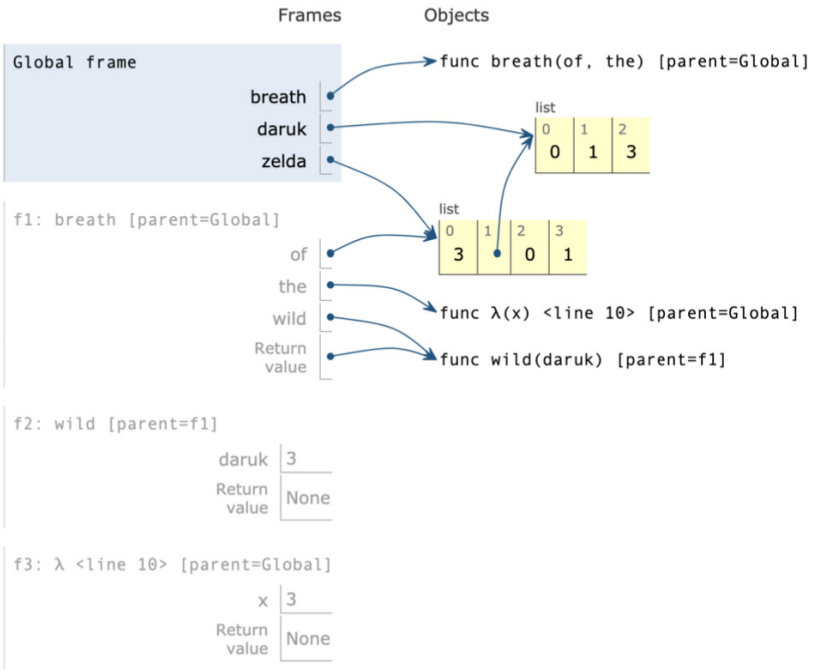
result = 2
low_r = 3
high_r = 5

Fall 2021 Q2



a → 3
b → parent = f3
c → f3
d → 2021
e → 3

Summer 2021 Q1



Click here to open the diagram in a new window

In this series of questions, you'll fill in the blanks of the program that follows so that its execution matches the environment diagram.

```
def breath(of, the):
    def wild(daruk):
        of.extend(______(a)______)
        ____(b)_____(____(c)_______)
    return wild


daruk = [0, 1]
zelda = [3, daruk]
breath(zelda, _________(d)_________)(3)
```