1. (10 points) Exeggcute

For each of the expressions in the table below, write the output displayed by the interactive Python interpreter when the expression is evaluated. The output may have multiple lines. If an error occurs, write "Error", but include all output displayed before the error. If a function value is displayed, write "Function".

The first two rows have been provided as examples.

Recall: The interactive interpreter displays the value of a successfully evaluated expression, unless it is `None`.

Assume that you have started `python3` and executed the following statements:

```
equals = lambda a, b: a == b
nemo = lambda n: lambda: print(n)
ray = nemo(1)

def if_function(f, g, h):
    if h:
        f()
    elif h:
        f(f())
    else:
        print(5 or 6)
    g()

def dory():
    print('fish')
    return lambda: 1/0
```

Expression	Interactive Output
`pow(2, 3)`	8
`print(4, 5) + 1`	4 5 Error
`equals(3==4, equals(5, equals(5, 5)))`	
`print(print(print(2)), print(3))`	
`print(nemo(print(5))())`	
`if_function(nemo(3), dory, 2)`	
`if_function(dory, nemo(2), ray())`	

1. (10 points) Exeggcute

For each of the expressions in the table below, write the output displayed by the interactive Python interpreter when the expression is evaluated. The output may have multiple lines. If an error occurs, write "Error", but include all output displayed before the error. If a function value is displayed, write "Function".

The first two rows have been provided as examples.

Recall: The interactive interpreter displays the value of a successfully evaluated expression, unless it is `None`.

Assume that you have started `python3` and executed the following statements:

```
equals = lambda a, b: a == b
nemo = lambda n: lambda: print(n)
ray = nemo(1)

def if_function(f, g, h):
    if h:
        f()
    elif h:
        f(f())
    else:
        print(5 or 6)
    g()

def dory():
    print('fish')
    return lambda: 1/0
```

Expression	Interactive Output
`pow(2, 3)`	8
`print(4, 5) + 1`	4 5 Error
`equals(3==4, equals(5, equals(5, 5)))`	True
`print(print(print(2)), print(3))`	2 None 3 None None
`print(nemo(print(5))())`	5 None None
`if_function(nemo(3), dory, 2)`	3 fish
`if_function(dory, nemo(2), ray())`	1 5 2

3. (9 points) Countizard

(a) (6 pt) Implement `counter`, which takes a non-negative single-digit integer `d`. It returns a function `count` that takes a non-negative integer `n` and returns the number of times that `d` appears as a digit in `n`. You may not use recursive calls or any features of Python not yet covered in the course.

```
def counter(d):
    """Return a function of N that returns the number of times D appears in N.

    >>> counter(8)(8018)
    2
    >>> counter(0)(2016)
    1
    >>> counter(0)(0)
    0
    """
    def count(_____):
        k = 0
        while _____:
            _____, last = _____, n % 10
            if _____:
                _____
        _____
    _____
```

3. (9 points) Countizard

(a) (6 pt) Implement `counter`, which takes a non-negative single-digit integer `d`. It returns a function `count` that takes a non-negative integer `n` and returns the number of times that `d` appears as a digit in `n`. You may not use recursive calls or any features of Python not yet covered in the course.

```
def counter(d):
    """Return a function of N that returns the number of times D appears in N.

    >>> counter(8)(8018)
    2
    >>> counter(0)(2016)
    1
    >>> counter(0)(0)
    0
    """
    def count(n):

        k = 0

        while n > 0:


            n, last = n // 10, n % 10


            if last == d:


                k += 1


        return k


    return count
```

4. (6 points) Caterepeat

(a) (4 pt) Implement `repeat_sum`, which takes a one-argument function `f`, a value `x`, and a non-negative integer `n`. It returns the sum of $n+1$ terms. Each term, indexed by `k` starting at 0, is the result of applying `f` to `x` repeatedly `k` times. You may assign to only one name in each of the three assignment statements. You may not use recursive calls or any features of Python not yet covered in the course.

```
def repeat_sum(f, x, n):
    """Compute the following summation of N+1 terms, where the last term
    calls F N times: x + f(x) + f(f(x)) + f(f(f(x))) + ... + f(f(...f(x)))

    >>> repeat_sum(lambda x: x*x, 3, 0) # 3
    3
    >>> repeat_sum(lambda x: x*x, 3, 1) # 3 + 9
    12
    >>> repeat_sum(lambda x: x+2, 3, 4) # 3 + 5 + 7 + 9 + 11
    35
    """
    total, k = 0, 0

    while ____________________________________________________________________:

        _______________ = ____________________________________________________

        _______________ = ____________________________________________________

        _______________ = ____________________________________________________

    return total
```

4. (6 points) Caterepeat

(a) (4 pt) Implement `repeat_sum`, which takes a one-argument function `f`, a value `x`, and a non-negative integer `n`. It returns the sum of $n+1$ terms. Each term, indexed by `k` starting at 0, is the result of applying `f` to `x` repeatedly `k` times. You may assign to only one name in each of the three assignment statements. You may not use recursive calls or any features of Python not yet covered in the course.

```
def repeat_sum(f, x, n):
    """Compute the following summation of N+1 terms, where the last term
    calls F N times: x + f(x) + f(f(x)) + f(f(f(x))) + ... + f(f(...f(x)))

    >>> repeat_sum(lambda x: x*x, 3, 0) # 3
    3
    >>> repeat_sum(lambda x: x*x, 3, 1) # 3 + 9
    12
    >>> repeat_sum(lambda x: x+2, 3, 4) # 3 + 5 + 7 + 9 + 11
    35
    """
    total, k = 0, 0


    while k <= n:


        total = total + x


        x = f(x)


        k = k + 1


    return total
```

(b) (2 pt) Implement `sum_squares`, which takes a non-negative integer `n` and uses `repeat_sum` to return the sum of the squares of the first `n` positive integers. Assume `repeat_sum` is implemented correctly. You may use `pow`, which raises its first argument to the power of its second: `pow(9, 2)` is 81 and `pow(9, 0.5)` is 3.0.

```
def sum_squares(n):
    """Return the sum of the first N perfect squares.

    >>> sum_squares(0)
    0
    >>> sum_squares(3) # 1**2 + 2**2 + 3**2
    14
    >>> sum_squares(5) # 1**2 + 2**2 + 3**2 + 4**2 + 5**2
    55
    """

    f = ------------------------------------------------------------------------

    return repeat_sum(f, 0, n)
```

(b) (2 pt) Implement `sum_squares`, which takes a non-negative integer `n` and uses `repeat_sum` to return the sum of the squares of the first `n` positive integers. Assume `repeat_sum` is implemented correctly. You may use `pow`, which raises its first argument to the power of its second: `pow(9, 2)` is 81 and `pow(9, 0.5)` is 3.0.

```
from repeat_sol import *

def sum_squares(n):
    """Return the sum of the first N perfect squares.

    >>> sum_squares(0)
    0
    >>> sum_squares(3) # 1**2 + 2**2 + 3**2
    14
    >>> sum_squares(5) # 1**2 + 2**2 + 3**2 + 4**2 + 5**2
    55
    """

    f = lambda x: pow(round(pow(x, 0.5) + 1), 2)

    return repeat_sum(f, 0, n)
```

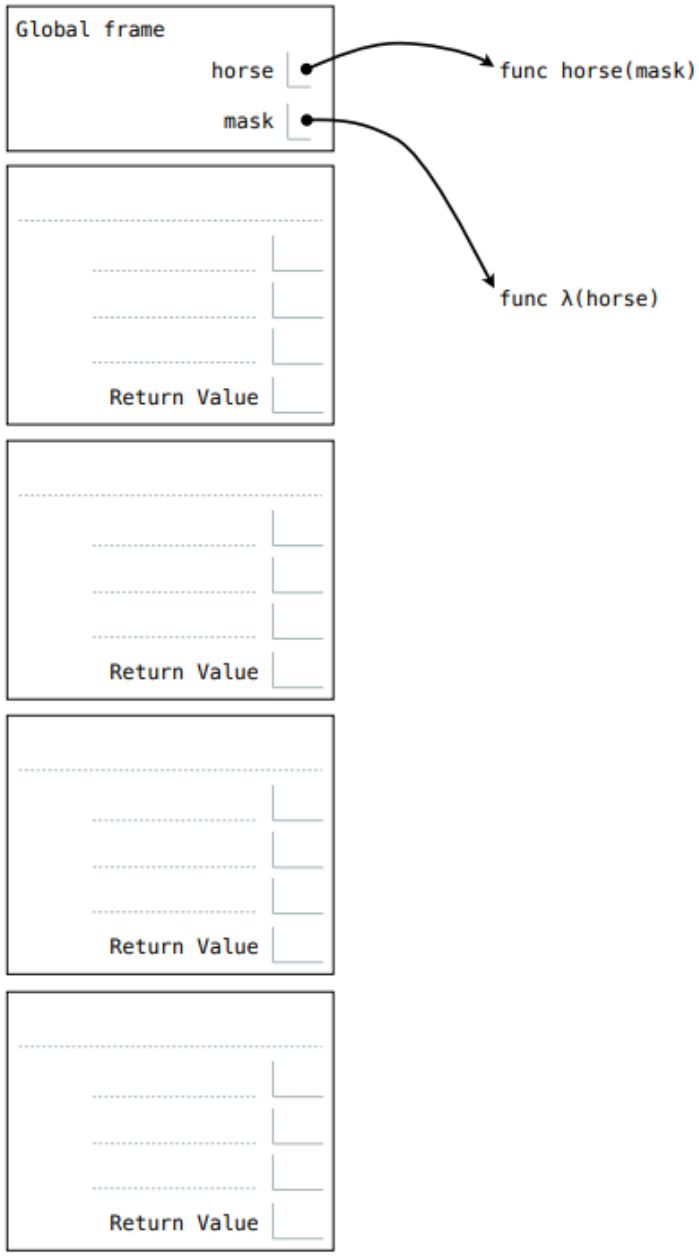
(a) (6 pt) Fill in the environment diagram that results from executing the code below until the entire program is finished, an error occurs, or all frames are filled. *You may not need to use all of the spaces or frames.* A complete answer will:

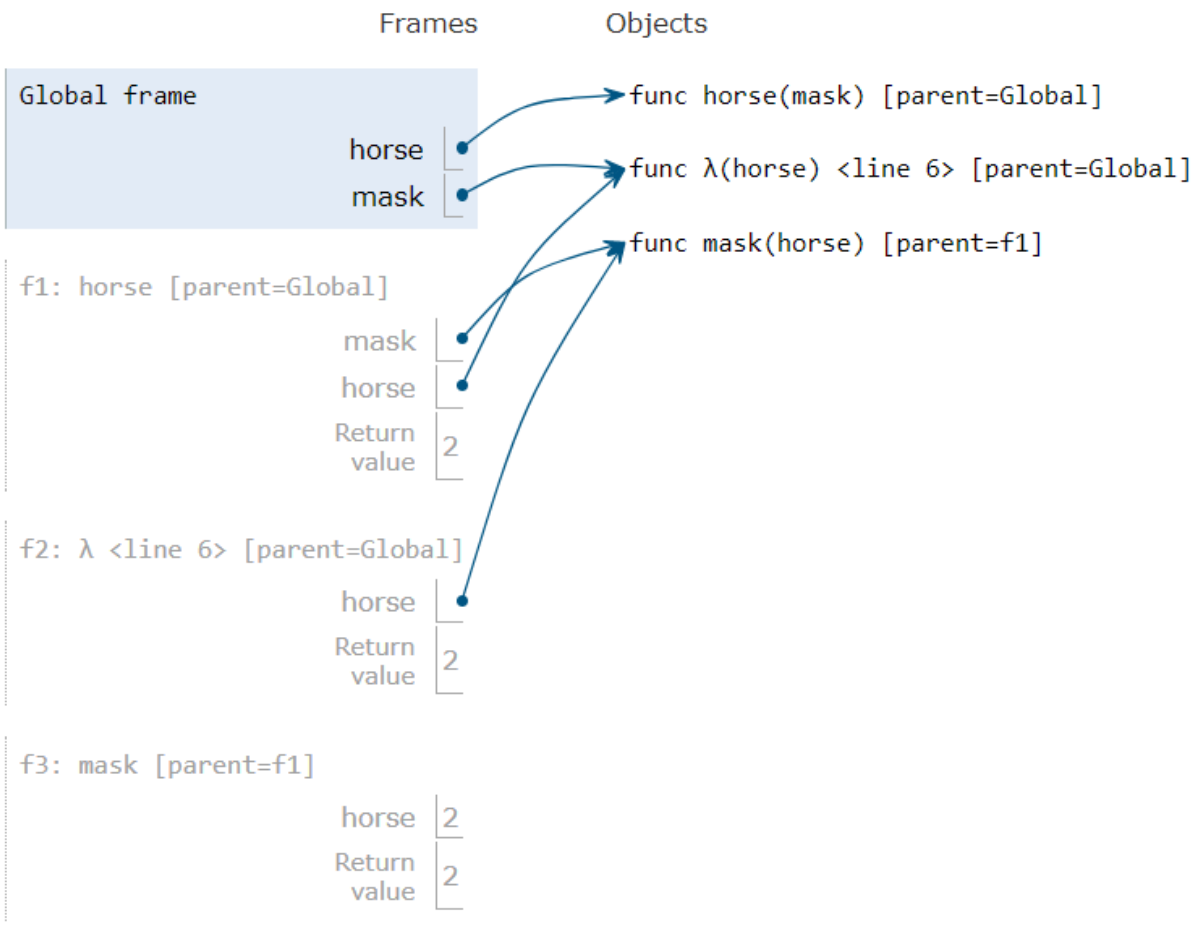
- Add all missing names, labels, and parent annotations to all local frames.
- Add all missing values created during execution.
- Show the return value for each local frame.

```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```




Frames
Objects
Global frame
horse
mask
func horse(mask) [parent=Global]
func λ(horse) <line 6> [parent=Global]
func mask(horse) [parent=f1]
f1: horse [parent=Global]
mask
horse
Return value
2
f2: λ <line 6> [parent=Global]
horse
Return value
2
f3: mask [parent=f1]
horse
2
Return value
2