1. Evaluators Gonna Evaluate

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". If a function is outputted, write "function".** Your answers must fit within the boxes provided. Work outside the boxes will not be graded.

Hint: No answer requires more than 6 lines. 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:

```
def anGenerator():
    x = 0
    while True:
        yield x
        x += 1

class GenIterator:
    def __init__(self):
    self.current = anGenerator()

    def __next__(self):
        return next(self.current)

    def __iter__(self):
        return self

class Flower:
    petals = True

    def __init__(self, colour):
        self.colour = colour

    def color(self):
        print("I'm colorful!")
```

```
class Tulip(Flower):
    season = "spring"

    def color(self):
        print(self.colour)


class Daffodil(Flower):

    def __init__(self, colour):
        self.colour = colour
        self.height = 0

    def color(self):
        print(self.colour)

    def grow(self, inches):
        self.height += inches

    def season(self):
        print("Season pushed back")
```

Expression	Interactive Output
`Flower.petals`	`True`
`Rose()`	`Error`
`tulip = Tulip("red")` `tulip.color()`	`red`

`daffodil = Daffodil("yellow")` `daffodil.color()`	`yellow`
`Flower.color(daffodil)`	`I'm colorful!`
`daffodil.petals`	`True`
`tulip.season = "early spring"` `print(Tulip.season, tulip.season)`	`spring early spring`
`tule = Tulip("purple")` `tule.season`	`"spring"`

`tulip = Tulip("blue")`
`Tulip.color(daffodil)`
`tulip.color(daffodil)`
	`yellow`
	`Error`

`tulip.height = 100`
`Daffodil.grow(tulip, 200)`
`Tulip.height`
	`Error`

`a = GenIterator()`
`for i in range(1, 6):`
`  print(next(a))`
	`0`
	`1`
	`2`
	`3`
	`4`

`for i in range(3):`
`  print(next(a))`
	`5`
	`6`
	`7`

`next(GenIterator())`	`0`

2. Some Tech Fame [Python Tutor Solution]

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.*

There are 20 blanks total you need to fill out!

A complete answer will:

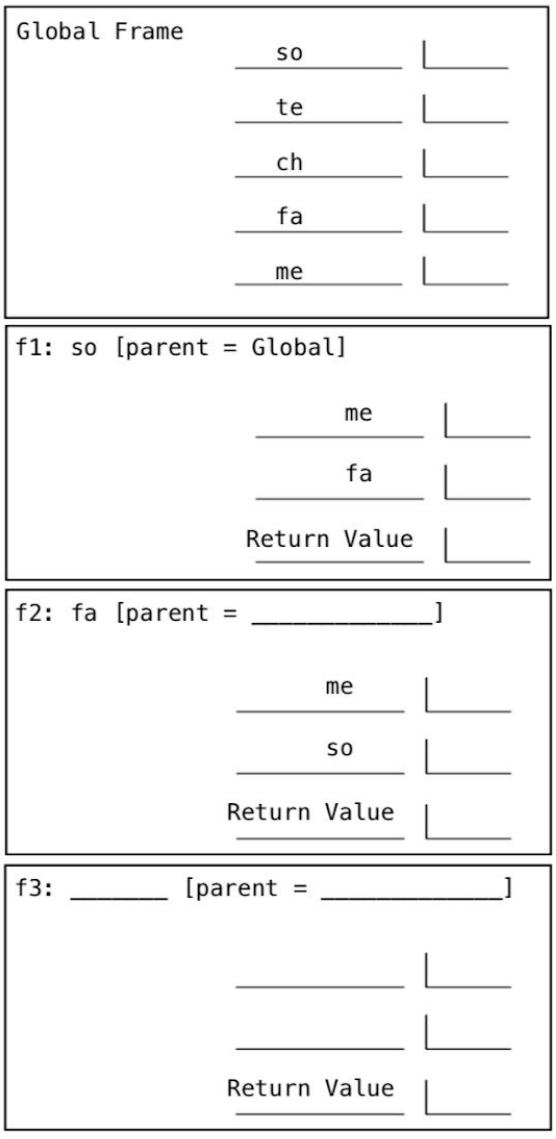
- Add all missing names and parent annotations to all local frames.
- Draw any necessary arrows to function names.
- Add all missing values created or referenced during execution.
- Show the return value for each local frame.

```
so = 5
te = 6
ch = [2, 4]

def so(me):
    me = 8
    def fa(me, so):
        so.append(me)
        return me + 1
    return fa

def fa(me, so):
    return [me] + so

te = so(te)(te, ch)
me = fa
me(['c', 'h'], ch)
```



func so(me) [parent = Global]

func fa(me, so) [parent = Global]

func fa(me, so) [parent = _____]

3. Warriors in 6

Answer the following questions given a table NBA containing players' results after a game of the following form:

```
Table: Players
name             | team           | college        | age
-------------------------------------------------------
DeMarcus Cousins | Golden State   | Kentucky       | 28
Kevin Durant     | Golden State   | Texas          | 30
James Harden     | Houston        | Arizona        | 29
Kawhi Leonard    | Toronto        | San Diego      | 27
Oski Bear        | Memphis        | California     | 22

Table: Stats
name             | minutes  | points   | rebounds | assists
-----------------------------------------------------------
DeMarcus Cousins | 0        | 0        | 0        | 0
Kevin Durant     | 28       | 35       | 5        | 3
James Harden     | 33       | 35       | 4        | 6
Kawhi Leonard    | 15       | 18       | 10       | 10
Oski Bear        | 24       | 101      | 39       | 31
```

A. What is the output of the following SQL query. Not all boxes will be necessary.

```
SELECT name, rebounds+assists, points FROM Stats WHERE points > minutes
ORDER BY points, name
```

Kawhi Leonard	20	18	
James Harden	10	35	
Kevin Durant	8	35	
Oski Bear	70	101	

B. Write a SQL query that retrieves the **name** of all players who had more rebounds than assists.

```
SELECT name FROM Stats WHERE rebounds > assists
```

C. Write a SQL query that retrieves the **name** and their **points per minute** for all players who played at least 1 minute.

```
SELECT name, points/minutes FROM Stats WHERE minutes != 0
```

D. Write a SQL query that retrieves the **name**, **college**, and **points** of all players.

```
SELECT stats.name, college, points FROM Players, Stats WHERE Players.name
= Stats.name
```

E. Get all unique pairs of player **names** who scored at least 30 points in a game.

```
SELECT a.name, b.name FROM Stats as a, Stats as b WHERE a.name < b.name
AND a.points + b.points > 60 ORDER BY a.name
```

4. Mutation

In a city of *N* people, represented by integers 1 to N, you are tasked in finding which person out of all of them is the mayor. Only one person can be mayor. You are given *pairs*, a list of 2-element lists in the form of [a,b]. Each pair denotes that person a *trusts* person b.

The mayor has two important properties:

1. The mayor is trusted by all of the other people.
2. The mayor trusts no one.

Complete the main function and helper functions below to return the integer that represents the mayor or -1 if the mayor does not exist. You can assume pairs is not an empty list and N > 1.

A. First, complete the `createTrusted` helper function.

```
def createTrusted(pairs):
    """ Returns a dictionary mapping a person to a list of people who
    trust them.

    >>> createTrusted([[1,3], [2,3], [3,1]])
    {3: [1, 2], 1: [3]}
    >>> createTrusted([[1,3], [1,4], [2,3], [2,4], [4,3]])
    {3: [1, 2, 4], 4: [1, 2]}
    """
    trusted = {}
    for pair in pairs:
        if pair[1] in trusted:
            trusted[pair[1]].append(pair[0])
        else:
            trusted[pair[1]] = [pair[0]]
    return trusted
```

B. Next, complete the `createTrusts` helper function.

```
def createTrusts(pairs):
    """ Returns a dictionary mapping a person to a list of people they
    trust.

    >>> createTrusts([[1,3], [2,3], [3,1]])
    {1: [3], 2: [3], 3: [1]}
    >>> createTrusts([[1,3], [1,4], [2,3], [2,4], [4,3]])
    {1: [3, 4], 2: [3, 4], 4: [3]}
    """
    trusts = {}
    for pair in pairs:
        if pair[0] in trusts:
            trusts[pair[0]].append(pair[1])
        else:
            trusts[pair[0]] = [pair[1]]
    return trusts
```

C. Finally, complete the findMayor function to solve our original problem. You may use `createTrusted` and `createTrusts` from above and can assume they work properly.

```
def findMayor(N, pairs):
    """
    >>> # 1 trusts 2, 2 doesn't trust anyone, so 2 is the mayor
    >>> findMayor(2, [[1,2]])
    2
    >>> # everyone trusts 3, but 3 trusts no one, so 3 is mayor
    >>> findMayor(3, [[1,3], [2,3]])
    3
    >>> # everyone trusts 3, but 3 trusts 1, so not mayor
    >>> findMayor(3, [[1,3], [2,3], [3,1]])
    -1
    >>> # No one is trusted by everyone, so no mayor
    >>> findMayor(3, [[1,2], [2,3]])
    -1
    >>> # everyone trusts 3, but 3 trusts no one, so 3 is mayor
    >>> findMayor(4, [[1,3], [1,4], [2,3], [2,4], [4,3]])
    3
    """
    trusted = createTrusted(pairs)
    trusts = createTrusts(pairs)

    most_trusted = []
    for key in trusted:
        if len(trusted[key]) == N-1:
            most_trusted += [key]

    for person in most_trusted:
        if person not in trusts:
            return person

    return -1
```

5. Perfect Numbers

A. Write a function that returns the list of all proper divisors of a number n.
Definition: x is a divisor of n if n % x == 0
Definition: x is a proper divisor if x is a divisor of n and x != n
In other words, a proper divisor of n is a number that evenly divides n and is not equal to n.

```
def get_proper_divisors(n):
    """
    >>> get_proper_divisors(1)
    [] # 1 is the only divisor of 1, but is not a proper divisor
    >>> get_proper_divisors(2)
    [1] # 1 and 2 are divisors of 2, but 1 is the only proper divisor
    >>> get_proper_divisors(3)
    [1]
    >>> get_proper_divisors(4)
    [1, 2]
    >>> get_proper_divisors(5)
    [1]
    >>> get_proper_divisors(6)
    [1, 2, 3]
    """
    divisor_lst = []
    for x in range(1, n):
        # We're not concerned with efficiency but can also just iterate
        # until the square root of n.
        if (n % x == 0):
            divisor_lst.append(i)
    return divisor_lst
```

B. Write a generator function perfect_nums() that continually yields successive perfect numbers. Perfect numbers are positive numbers that are equal to the sum of their proper divisors. You can assume that get_proper_divisors() is implemented correctly and may use it in this problem.

```
def perfect_nums():
    """
    >>> perfect_num_gen = perfect_nums()
    >>> next(perfect_num_gen)
    6 # 6 is the first perfect number because its proper divisors are 1,
        2, 3 which sum to itself
    >>> next(perfect_num_gen)
    28 #28 is the second perfect number because its proper divisors are
1, 2, 4, 7, 14 which sum to itself

    current_num = 0
    while True:
        current_num += 1
        proper_divisors = get_proper_divisors(current_num)
        if (sum(proper_divisors) == current_num):
            yield current_num
```

6. Time Is Money

Fill in the __next__ method in Timer and the pass_time method in KitchenCounter. A timer should step forward one second each time next is called. Once the timer runs out, you should print out a message that says the food is ready. KitchenCounter maintains a list of multiple timers; pass_time should step forward all of the timers by that much time. The timers should always be within one second of each other (i.e. increment all of the timers once before incrementing any timer twice.
TIP: Don't forget about StopIteration.

```
class Timer:
    """
    >>> a = Timer("Pete Zaroll", 2, "seconds")
    >>> [i for i in a]
    Pete Zaroll is ready!
    [1, 2]
    """
    # Maps a unit string to a multiplier that converts it to seconds
    unit2Seconds = {"seconds" : 1, "minutes" : 60, "hours" : 60*60,
"days": 24*60*60}
    def __init__(self, food, time, unit):
        self.food = food
        self.current = 1
        self.time = time * self.unit2Seconds[unit]

    def __iter__(self):
        return self

    def ready(self):
         print(self.food + " is ready!")

    def __next__(self):
        if self.current >= self.time:
            self.ready()
            raise StopIteration
        else:
            self.current += 1
            return self.current - 1
```

```
class KitchenCounter:
    """
    >>> a = Timer("Pete Zaroll", 15, "minutes")
    >>> b = Timer("Chim E Changa", 20.5, "minutes")
    >>> c = Timer("Pho Lah Phil", 12, "seconds")
    >>> k = KitchenCounter()
    >>> k.add_timer(a)
    >>> k.add_timer(b)
    >>> k.add_timer(c)
    >>> k.pass_time(10, "seconds")
    10 seconds passed
    >>> k.pass_time(2, "seconds")
    Pho Lah Phil is ready!
    2 seconds passed
    >>> k.pass_time(15, "minutes")
    Pete Zaroll is ready!
    15 minutes passed
    >>> k.pass_time(5.5, "minutes")
    Chim E Changa is ready!
    5.5 minutes passed
    """

    # Maps a unit string to a multiplier that converts it to seconds
    unit2Seconds = {"seconds" : 1, "minutes" : 60, "hours" : 3600,
"days": 86400}

    def __init__(self):
        self.timers = []

    def add_timer(self, timer):
        self.timers.append(timer)

    def pass_time(self, time, units):
        """
        Pass the appropriate amount of seconds on each timer,
        removing (lists have a remove method:), it once it's
        time has run out.
        """
        seconds = int(self.unit2Seconds[units]*time)

        for i in range(seconds):
            for timer in self.timers:
                try:
                    next(timer)
                except StopIteration:
```

```
                self.timers.remove(timer)
        print(str(time) + " " + str(units) + " passed")
```

7. Class Is in Session

Fill out this class to match the interactive outputs:

```
>>> andrew = Person("Andrew")
>>> andrew.say()
Hi I'm Andrew
>>> alex = TA("Alex")
>>> amir = Student("Amir", alex)
>>> amir.say()
Hi I'm Amir and I'm in Alex's lab
>>> alex.add_student(amir)
>>> alex.add_student(Student("Jessica", alex))
>>> alex.say()
Hi I'm Alex and my students are Amir Jessica
>>> alex.add_student(Student(“Gerald”, alex))
>>> alex.say()
Hi I'm Alex and my students are Amir Jessica Gerald
```

```
class Person:
    def __init__(self, name):
        self.name = name

    def say(self):
        print("Hi I'm " + self.name)

class Student(Person):
    def __init__(self, name, ta):
        super.__init__(self, name)
        self.ta = ta

    def say(self):
        print("Hi I'm " + self.name + " in " + self.ta.name + "'s lab")

class TA(Person):
    def __init__(self, name):
        super.__init__(self, name)
        self.students = []

    def add_student(self, student):
        self.students.append(student)


    def say(self):
        student_string = “”
        for student in self.students:
            student_string += student.name + “ “
        print(“Hi I’m “ + self.name + “ and my students are “ + student_string)
```