INSTRUCTIONS

- You have 3 hours to complete the exam. Put your name and SID on every page.
- The exam is closed book; no resources are allowed except two 8.5” × 11” cheat sheets and the official CS 88 final reference sheet (attached to the back of the exam). Remove the reference sheet before turning in exam.
- Mark your answers on the exam itself. We will not grade answers written on scratch paper. Check that you have 8 double-sided pages (including cover page) for 7 problems.

<table>
<thead>
<tr>
<th>Last name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>First name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Student ID number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Berkeley email [@berkeley.edu]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name of the person to your left</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name of the person to your right</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

All the work on this exam is my own. (please sign)

POLICIES & CLARIFICATIONS

- You may use built-in Python functions that do not require import, such as `min`, `max`, `pow`, and `abs`. You may not use functions defined on your study guide unless clearly specified in the question.
- For fill-in-the blank coding problems, we will only grade work written in the provided blanks. You may only write one Python statement per blank line, and it must be indented to the level that the blank is indented. Your solution must fit within the number of lines provided, but may not require all of the lines.
- Unless otherwise specified, you are allowed to reference functions defined in previous parts of the same question.
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:

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

class Tulip(Flower):
    season = “spring”
    def color(self):
        print(self.colour)

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

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

    def __iter__(self):
        return self
class Daffodil(Flower):
    def __init__(self, colour):
        self.colour = colour
        self.height = 0
    def color(self):
        print(self.colour)
class Flower:
    petals = True

    def __init__(self, colour):
        self.colour = colour
    def color(self):
        print(“I’m colorful!”)
def grow(self, inches):
    self.height += inches
    def season(self):
        print(“Season pushed back”)
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flower.petal</td>
<td>True</td>
</tr>
<tr>
<td>Rose()</td>
<td>Error</td>
</tr>
<tr>
<td>tulip = Tulip(“red”)</td>
<td></td>
</tr>
<tr>
<td>tulip.color()</td>
<td></td>
</tr>
</tbody>
</table>
```python
daффодил = Даводил("yellow")
daффодил.color()

Цветок.color(драводил)

dраводил.лепестки

tulip.season = "early spring"
print(Тюльпан.season, tulip.season)

tул = Тюльпан("purple")
tул.season

tulip = Тюльпан("blue")
Тюльпан.color(драводил)
tulip.color(драводил)

tulip.height = 100
Даводил.рост(tulip, 200)
Тюльпан.height

a = Генератор()
for i in range(1, 6):
    print(next(a))

for i in range(3):
    print(next(a))

next(Генератор())
```
2. Some Tech Fame

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.

```python
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 SQL questions given tables **Players** and **Stats** of the following form:

**Table: Players**

<table>
<thead>
<tr>
<th>name</th>
<th>team</th>
<th>college</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeMarcus Cousins</td>
<td>Golden State</td>
<td>Kentucky</td>
<td>28</td>
</tr>
<tr>
<td>Kevin Durant</td>
<td>Golden State</td>
<td>Texas</td>
<td>30</td>
</tr>
<tr>
<td>James Harden</td>
<td>Houston</td>
<td>Arizona</td>
<td>29</td>
</tr>
<tr>
<td>Kawhi Leonard</td>
<td>Toronto</td>
<td>San Diego</td>
<td>27</td>
</tr>
<tr>
<td>Oski Bear</td>
<td>Memphis</td>
<td>California</td>
<td>22</td>
</tr>
</tbody>
</table>

**Table: Stats**

<table>
<thead>
<tr>
<th>name</th>
<th>minutes</th>
<th>points</th>
<th>rebounds</th>
<th>assists</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeMarcus Cousins</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Kevin Durant</td>
<td>28</td>
<td>35</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>James Harden</td>
<td>33</td>
<td>35</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Kawhi Leonard</td>
<td>15</td>
<td>18</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Oski Bear</td>
<td>24</td>
<td>101</td>
<td>39</td>
<td>31</td>
</tr>
</tbody>
</table>

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

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

D. Write a SQL query that retrieves the **name**, **college**, and **points** of all players. Note: your query output should NOT repeat any rows.

E. Write a SQL query that retrieves all **unique pairs** of player **names** if the sum of the 2 players’ points is greater than 60. Order the pair of names in each row by alphabetical order, and order the rows in alphabetical order by the first player in the pair. Here is the expected output:

<table>
<thead>
<tr>
<th>DeMarcus Cousins</th>
<th>Oski Bear</th>
</tr>
</thead>
<tbody>
<tr>
<td>James Harden</td>
<td>Kevin Durant</td>
</tr>
<tr>
<td>James Harden</td>
<td>Oski Bear</td>
</tr>
<tr>
<td>Kawhi Leonard</td>
<td>Oski Bear</td>
</tr>
<tr>
<td>Kevin Durant</td>
<td>Oski Bear</td>
</tr>
</tbody>
</table>
4. Find the Mayor

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 \( \text{pairs} \), a list of 2-element lists in the form of \([a, b]\). Each pair \([a, b]\) 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 \( \text{pairs} \) is not an empty list and \( N > 1 \).

A. First, complete the \texttt{createTrusted} helper function.

```python
def createTrusted(pairs):
    ""
    Returns a dictionary mapping a person to a list of people who trust them. The order of the list of people does not matter.
    ""

>>> 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 = {}

-----------------------------------------------
-----------------------------------------------
-----------------------------------------------
-----------------------------------------------
-----------------------------------------------
-----------------------------------------------
-----------------------------------------------

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. The order of the list of people does not matter.
    """

    >>> 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 = {}

    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.

```python
def findMayor(N, pairs):
    """ Return the integer representing the mayor with the properties:
    1. The mayor is trusted by all of the other people.
    2. The mayor trusts no one.
    Return -1 if no such mayor exists.
    """

    >>> findMayor(2, [[1,2]])
    2 # 1 trusts 2, 2 doesn’t trust anyone, so 2 is the mayor
    >>> findMayor(3, [[1,3], [2,3]])
    3 # everyone trusts 3, but 3 trusts no one, so 3 is mayor
    >>> findMayor(3, [[1,3], [2,3], [3,1]])
    -1 # everyone trusts 3, but 3 trusts 1, so not mayor
    >>> findMayor(3, [[1,2], [2,3]])
    -1 # No one is trusted by everyone, so no mayor
    >>> findMayor(4, [[1,3], [1,4], [2,3], [2,4], [4,3]])
    3 # everyone trusts 3, but 3 trusts no one, so 3 is mayor
    """

    trusted = ________________________________
    trusts = ________________________________
    """

    return -1
```
5. Perfect Numbers

A perfect number is a positive integer that is equal to the sum of its proper positive divisors, that is, the sum of its positive divisors excluding the number itself.

A. First, write a function that returns the list of all proper divisors of a number n. A proper divisor of n is a positive integer that evenly divides n and is not equal to n. Assume n is a positive integer and we only want divisors that are also positive integers.

Definition: x is a divisor of n if n % x == 0
Definition: x is a proper divisor of n if x is a divisor of n and x != 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]
    
    ---
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.

```python
def perfect_nums():
    """Generate each successive perfect number.
    >>> 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
    """
```

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 timers; pass_time should step forward all of the timers by the amount of seconds specified by the time and unit arguments. 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 Error.

class Timer:
    """
    >>> a = Timer("Pete Zaroll", 2, "seconds")
    >>> b = [i for i in a]
    Pete Zaroll is ready!
    """
    # Maps a unit string to a multiplier that converts it to seconds
    unit2Seconds = {"seconds" : 1, "minutes" : 60, "hours" : 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):
        return self

Name and SID: ________________________________
```python
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_timers([a, b, c])
    >>> k.pass_time(12, "seconds")
    Pho Lah Phil is ready!
    12 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
    
    unit2Seconds = {"seconds" : 1, "minutes" : 60, "hours" : 60*60}
    def __init__(self):
        self.timers = []

    def add_timers(self, timers):
        self.timers += timers

    def pass_time(self, self, time, units):
        
        """Increment each timer in self.timers by the appropriate amount of
        seconds. Remove any timer from the list of timers once its time has run
        out. Hint: lists have a remove method. Hint: StopIteration
        """

        seconds = int(self.unit2Seconds[units]*time)

        print(str(time) + " " + str(units) + " passed")
```
7. Class Is in Session

Implement the 3 classes to match the interactive outputs below:

```python
$ python3
>>> 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 ________________________________:

    def __init__(self, name, ta):
        super.__init__(_________, _________)

                           ________________________________
                           ________________________________
                           ________________________________

    def say(self):

                           ________________________________

        print(______________________________)
```
class ______________________________:

def __init__(________, __________):

    ______________________________
    ______________________________

def add_student(self, student):

    ______________________________
    ______________________________
    ______________________________
    ______________________________

def say(self):

    ______________________________
    ______________________________
    ______________________________
    ______________________________
    ______________________________
    print(__________________________)
Numeric types in Python:

- **type(2)**
  - Represents integers exactly
- **type(1.5)**
  - Represents real numbers approximately
- **type(1+1j)**
  - Represents complex numbers

Rational implementation using functions:

```python
def select(name):
    if name == 'n':
        return n
    elif name == 'd':
        return d
    return select

def numerator(x):
    return x('n')

def denominator(x):
    return x('d')
```

Lists:

- **len(digits)**
  - Evaluates to 8
- **digits[3]**
  - Evaluates to 3
- **digits[2] + digits[2]**
  - Evaluates to 4

Strings as sequences:

- `'k'` is a string

Strings as sequences:

- `str` and `repr` are both polymorphic; they apply to any object in a combination that evaluates to a list using `__iter__`.

Equality:

- `b = a` and `b == a` are both `True`
- `a.append(20)` and `b.append(20)` are both `True`
- `a` and `b` are both `[10, 20]`
- `a == b` is `False`

List & dictionary mutation:

- `nums = {'T': 1.0, 'V': 5, 'X': 10}`
  - `nums['X']` evaluates to 10
- `nums['T'] = 1`
- `nums['L'] = 50`
- `nums['X'] = 10`
- `nums['L'] = 50`
- `nums[10] = 5`

Strings as sequences:

- `city = 'Berkeley'`
  - `len(city)` evaluates to 8
- `city[3]` evaluates to `'k'`
  - `str('here')` in `Where's Waldo?` evaluates to `True`
  - `234 in [1, 2, 3, 4, 5]` evaluates to `False`
  - `[2, 3, 4] in [1, 2, 3, 4]` evaluates to `False`
- `nums.get('T', 0)` evaluates to 0
- `nums.keys()` evaluates to `{'T': 1.0, 'V': 5, 'X': 10}`
- `nums.items()` evaluates to `[(10, 20), (5, 25)]`
- `nums.get('A', 0)` evaluates to 0
- `nums.get('V', 5)` evaluates to 5
- `[x for x in range(3, 6)]` evaluates to `[2, 3, 4, 5]`

Slicing:

- `digits[0:2]` evaluates to `[1, 8]`
- `digits[1:]` evaluates to `[8, 2, 8]`
- `Slicing creates a new object`
Tree data abstraction:

A tree has a root node and a sequence of branches; each branch is a tree

```python
def tree(root, branches=[]):
    for branch in branches:
        assert is_tree(branch)
    return [root] + list(branches)
def root(tree):
    return tree[0]
def branches(tree):
    return tree[1:] def is_tree(tree):
    if type(tree) == list or len(tree) < 1:
        return False
    for branch in branches(tree):
        if not is_tree(branch):
            return False
    return True
def is_leaf(tree):
    return not branches(tree)
def leaves(tree):
    """The leaf values in tree.""
    return [leaf for branch in branches(tree)
            for leaf in leaves(branch)]
def fib_tree(n):
    if n == 0 or n == 1:
        return tree(n)
    left = fib_tree(n-2)
    right = fib_tree(n-1)
    fib_n = root(left) + root(right)
    return tree(fib_n, [left, right])
def head(s):
    if s == [] or s == []
        return False
    for branch in branches(s):
        if not is_tree(branch):
            return False
    return True
```

Python object system:

Idea: All bank accounts have a balance and an account holder; 
The Account class should add those attributes to each of its instances

```python
>>> a = Account('Jim')
>>> a.balance = 0
>>> a.balance
0
```

When a class is called:

1. A new instance of that class is created:
   - balance: 0  holder: 'Jim'
2. The `__init__` method of the class is called with the new object as its first argument (named self), along with any additional arguments provided in the call expression.

```python
class Account:
    def __init__(self, account_holder, balance=0):
        self.balance = balance
        self.holder = account_holder
def deposit(self, amount):
    self.balance += amount
    return self
```

Method invocation:

One object before the dot and other arguments within parentheses

```python
>>> Account.deposit(a, 5)
```

Assignment statements with a dot expression on their left-hand side affect the object of the dot expression
- If the object is an instance, then assignment sets an instance attribute
- If the object is a class, then assignment sets a class attribute

```python
>>> account = Account('Tom')
>>> account.holder = 'Jim'
```

To look up a name in a class:

1. If it names an attribute in the class, return the attribute value.
2. Otherwise, look up the name in the base class, if there is one.

```python
>>> ch = CheckingAccount('Tom')  # Calls Account.__init__
>>> ch.holder  # Found in CheckingAccount
'Tom'
```

Sequence abstraction special names:

- `__getitem__` Element selection []
- `__len__` Built-in len function

```python
class Link:
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest
def __getitem__(self, i):
    if i == 0:
        return first
    return _len_(self)[i-1]
def __len__(self):
    return 1 + _len_(self)
def _repr_(self):
    if self:
        rest_str = '', repr(self.rest)
    else:
        rest_str = ''
    return 'Link(%s).format(self.first, rest_str)
def extend_link(s, t):
    """Return a Link with the elements of s followed by those of t.""
    if s is Link.empty:
        return t
    else:
        return Link(s.first, extend_link(s.rest, t))
def map_link(f, s):
    if s is Link.empty:
        return s
    else:
        return Link(f(s.first), map_link(f, s.rest))
```