Create Rubric

100 points

List View | Grid View

Create your rubric now or come back to it later. You can also make edits to your rubric while grading.

Q1 Conceptual

9 points

Q1.1

3 points

Rubric

Suppose the following code is run sequentially:

```
def mystery(func, lst):
    if lst == Link.empty:
        return Link.empty
    else:
        return Link(func(lst.first), mystery(func, lst.r

lst = Link(1, Link(2, Link(3, Link(4))))
mystery_lst = mystery(lambda x: x * 2, lst)
```

For reference, here is the `Link` class definition:

```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest
```

What is the value of mystery_lst?

Link(2, Link(4, Link(6, Link(8))))

Link(4, Link(8))

Link(3, Link(4, Link(5, Link(6))))

Link(2, Link(6))

Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1 +3.0
Correct

2 +0.0
Incorrect

Add Rubric Item | Create Group | Ir

Q1.2

2 points

Rubric

Which built-in Python function does the function `mystery` behave the same as?

filter

reduce

map

max

⚠ Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1 +2.0

Correct

2 +0.0

Incorrect

+ Add Rubric Item | Create Group | Ir

Q1.3

2 points

Rubric

Suppose the following code is run sequentially:

```
kiwi = [1,9,8,6]
grape = [9,2,3,6]
```

Which of the following options would result in the output `[1, 9, 8, 6, 9, 2, 3, 6]`? Select all that apply.

Assume that each choice is independent of each other AND don't affect each other.

```
# (Choice A)
>>> result = kiwi + grape
>>> result

# (Choice B)
>>> result = grape + kiwi
>>> result

# (Choice C)
>>> kiwi.append(grape)
>>> kiwi

# (Choice D)
>>> kiwi.extend(grape)
>>> kiwi

# (Choice E)
>>> kiwi += grape
>>> kiwi

# (Choice F)
>>> grape += kiwi
>>> grape
```

⚠ Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1 +2.0

Correct

2 +0.0

Incorrect

+ Add Rubric Item | Create Group | Ir

- [✓ Choice A]
- Choice B
- Choice C
- [✓ Choice D]
- [✓ Choice E]
- Choice F

Q1.4

2 points

Rubric

Suppose we have two tables in SQL, `table_a` with `n` rows and `table_b` with `m` rows. Suppose we perform the following join query:

```
select * from table_a, table_b;
```

How many rows will this query output?

- m
- n
- m - n
- m + n
- m * n

⚠ Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1	+2.0	Correct
2	+0.0	Incorrect

+ Add Rubric Item | Create Group | Ir

Q2 Environment Diagrams

10 points

Fill in the blanks to complete the environment diagram. All the code used is as follows, and the code runs to completion.

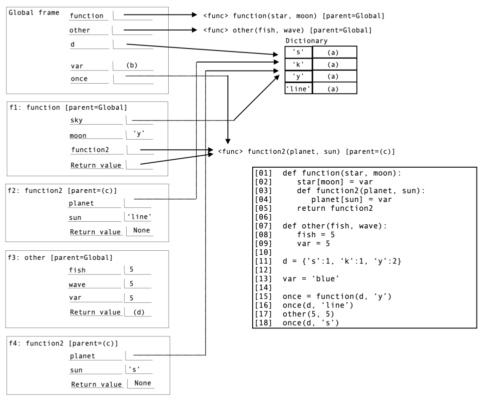
```
def function(star, moon):
    star[moon] = var
    def function2(planet, sun):
        planet[sun] = var
    return function2

def other(fish, wave):
    fish = 5
    var = 5

d = {'s' : 1, 'k' : 1, 'y': 2}

var = 'blue'

once = function(d, 'y')
once(d, 'line')
other(5, 5)
once(d, 's')
```



Global frame
function
other
d
var
(b)
once
<func> function(star, moon) [parent=Global]
<func> other(fish, wave) [parent=Global]
Dictionary
's'
(a)
'k'
(a)
'y'
(a)
'line'
(a)
f1: function [parent=Global]
sky
moon
'y'
function2
Return value
<func> function2(planet, sun) [parent=(c)]
f2: function2 [parent=(c)]
planet
sun
'line'
Return value
None
f3: other [parent=Global]
fish
5
wave
5
var
5
Return value
(d)
f4: function2 [parent=(c)]
planet
sun
's'
Return value
None

```
def function(star, moon):
   star[moon] = var
   def function2(planet, sun):
      planet[sun] = var
   return function2

def other(fish, wave):
   fish = 5
   var = 5

d = {'s':1, 'k':1, 'y':2}

var = 'blue'

once = function(d, 'y')
once(d, 'line')
other(5, 5)
once(d, 's')
```

Q2.1

3 points

⚙ Rubric

After the above has executed, what would be the output of `d.values()`? Provide the values in the format of a four element list. (Blank a)

['blue', 1, 'blue', 'blue']

⚠ Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1 **+3.0**
Correct

2 **+0.0**
Incorrect

3 **+3.0**
Correct

+ Add Rubric Item | Create Group | Ir

Q2.2

2 points

⚙ Rubric

What is the value of the variable `var` in the global frame? (Blank b)

'blue'

⚠ Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1 +2.0
Correct

2 +0.0
Incorrect

+ Add Rubric Item | Create Group | Ir

Q2.3

3 points

Rubric

What is the parent frame of the function `function2`? (Blank c)

global
f1
f2
f3
f4

⚠ Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1 +3.0
Correct

2 +0.0
Incorrect

+ Add Rubric Item | Create Group | Ir

Q2.4

2 points

Rubric

What is the return value of frame 3 (f3)? (Blank d)

None

⚠ Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1 +2.0
Correct

2 +0.0
Incorrect

+ Add Rubric Item | Create Group | Ir

Q3 What Would Python Do?

10 points

For each expression 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 any lines are displayed before the error, include those in your output). If a function is returned, write "Function". If the value "None" is returned, write "None".

NOTE: Assume each part is executed in order. Previous lines DO impact the current expression. (i.e., part B assumes part A was executed, as so on.)

```
def changer(lst, f, g):
    filtered = list(filter(f, lst))
    if True in filtered:
        return list(map(g, lst))
    else:
        return reduce(g, lst)

brat = [1, 2, 3, 4, 5, 6, 7, 8]
charlie = lambda x: x % 2==0
xcx = lambda a, b: a + b
```

Q3.1

2 points

Rubric !

```
>>> changer(brat, charlie, xcx)
```

36

Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1	+2.0	Correct
2	+0.0	Incorrect

+ Add Rubric Item | Create Group | Ir

Q3.2

2 points

Rubric

```
>>> xcx = lambda a: "talk"
>>> changer(brat, charlie, xcx)
```

Error

"talktalktalktalktalktalktalktalk"

"talktalktalktalk"

"talk"

Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1 +2.0

Correct

2 +0.0

Incorrect

Add Rubric Item | Create Group | Ir

Q3.3

2 points

Rubric

```
>>> apple = lambda y: y
>>> brat = [True, 0, '360', {}, False, 365]
>>> changer(brat, apple, lambda c, d: c * d)
```

Error

Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1 +2.0

Correct

2 +0.0

Incorrect

Add Rubric Item | Create Group | Ir

Q3.4

2 points

Rubric

Assume that this is a new environment; every variable defined above is no longer accessible

```
>>> [[x for x in range(i)] for i in range(5) if i % 2 ==
```

Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1 +2.0

`[[0], [0, 1, 2]]`

[[0], [0, 1, 2]]

2 +2.0

`[[0],[0,1,2]]`

3 +2.0

`[[0], [0,1,2]]`

4 +2.0

`[[0], [0,1,2]]`

5 +0.0

Incorrect

+ Add Rubric Item | Create Group | Ir

Q3.5

2 points

Rubric S

Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

Assume that this is a new environment; every variable defined above is no longer accessible

```
>>> lst = [1, 2, 3]
>>> lst.append(lst.append(4))
>>> lst
```

[1, 2, 3, 4, None]

1 +2.0

`[1, 2, 3, 4, None]`

2 +0.0

Incorrect

+ Add Rubric Item | Create Group | Ir

Q4 Debugging

10 points

Data C88C staff want to determine how effective discussion sections are. They want to see if there is a noticeable difference between the quiz scores of students from different discussion sections.

They have written the function `most_points` below to help determine this information. `most_points` takes in a dictionary `sections` that maps discussion TAs to a list of students in their section and takes in a dictionary `scores` that maps each student to a list of their quiz scores.

`most_points` should return a dictionary that maps each TA to the total sum of points their students scored across all quizzes.

```
>>> discussion_sections = {
  'mia' : ['dan', 'serena', 'jenny'],
  'satleen' : ['chuck', 'blaire', 'nate']
}

>>> quiz_scores = {
  'blaire' : [9, 9, 8], 'chuck' : [6, 5, 9],
  'dan' : [7, 7, 2], 'jenny' : [4, 6, 2],
  'nate' : [5, 7, 6], 'serena' : [9, 10, 0]
}
>>> most_points(discussion_sections, quiz_scores)
{'mia' : 47, 'satleen': 64}
```

Here is an incorrect implementation of `most_points`:

```
1 def most_points(sections, scores):
2     points = []
3     for key in sections:
4         score_sum = 0
5         for i in value:
6             score_sum += sum(scores[i])
7         points[key] = score_sum
8     return points
```

Unfortunately, the Data C88C staff wrote this code after hours of quiz grading and overlooked some bugs. Help them work through the bugs and correct their code.

Q4.1

3 points

Rubric

I claim that calling the `most_points()` function as it is currently written will result in an error. On what line will the function error?

⚠ Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1 **+3.0**

Correct

Line 2

Line 3

Line 5

Line 7

2 +0.0

Incorrect

+ Add Rubric Item | Create Group | Ir

Q4.2

3 points

Rubric

The current code may have a bug on line 2. Rewrite line 2 to avoid this bug or write "No bug" if there is no bug.

points = {}

Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1 +3.0

Correct: points = {}

2 +0.0

Incorrect

+ Add Rubric Item | Create Group | Ir

Q4.3

4 points

Rubric

Ignoring all other bugs, lines 3-6 can operate as intended by changing just one line. Ignore all bugs outside of lines 3-6. Consider each edit independently.

- line 3. Replace `for key in sections:` with `for key, value in sections.items():`
- line 3. Replace `for key in sections:` with `for value in sections.values():`
- line 6. Replace `score_sum += sum(scores[i])` with `score_sum += sum(value)`

Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1 +4.0

Correct

2 +0.0

Incorrect

+ Add Rubric Item | Create Group | Ir

Q5 Object Oriented Programming

14 points

Consider the following code modeling `Person` eating at `Restaurants`:

```
class Restaurant:
    def __init__(self, food_strength):
        self.food_strength = food_strength

    def serve_customer(self, person):
        if person.hunger > 0:
            person.eat(self.food_strength)
        if person.hunger > 0:
            return f"wow {person.name} is still hungry"
        else:
            return f"{person.name} is full"

class Person:
    def __init__(self, name):
        self.name = name
        # a person is "full" if their hunger is <= 0
        # a full person will not eat more
        self.hunger = 10

    def eat(self, food_strength):
        self.hunger -= food_strength
        return self.hunger

# Example: Alice eats at the golden_bear_cafe Restaurant
>>> alice = Person("Alice")
>>> f"Alice hunger: {alice.hunger}"
Alice hunger: 10
>>> golden_bear_cafe = Restaurant(8)
>>> golden_bear_cafe.serve_customer(alice)
wow Alice is still hungry
>>> f"Alice hunger: {alice.hunger}"
Alice hunger: 2
>>> golden_bear_cafe.serve_customer(alice)
Alice is full
>>> f"Alice hunger: {alice.hunger}"
Alice hunger: -6
# Full people don't eat more
>>> golden_bear_cafe.serve_customer(alice)
Alice is full
>>> f"Alice hunger: {alice.hunger}"
Alice hunger: -6
```

Q5.1

3 points

⚙ Rubric

⚠ Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

Suppose I want to create a `Student` class that is just like a `Person`, but can literally eat as much as they want (their `hunger` never drops). In other words, a hungry `Student` can eat forever:

```
>>> top_dog = Restaurant(8)
>>> cecilia = Student("Cecilia")
>>> cecilia.hunger
```

1 +3.0

Correct

2 +0.0

```
10
>>> top_dog.serve_customer(cecilia)
wow Cecilia is still hungry
>>> top_dog.serve_customer(cecilia)
wow Cecilia is still hungry
>>> top_dog.serve_customer(cecilia)
wow Cecilia is still hungry
>>> cecilia.hunger
10
```

Incorrect

+ Add Rubric Item | Create Group | Ir

Which implementation correctly implements the desired behavior?

```
# Choice A
class Student(Person):
    def eat(self):
        return self.hunger

# Choice B
class Student:
    def eat(self, food_strength):
        return self.hunger

# Choice C
class Student(Person):
    def eat(self, food_strength):
        return self.hunger

# Choice D
class Student(Person):
    def eat(self, food_strength):
        super().eat(food_strength)
        return self.hunger
```

Choice A

Choice B

Choice C

Choice D

Q5.2

3 points

Rubric

Suppose I wanted each `Restaurant` to keep track of all customers they have ever served, via a `customer_history`:

⚠ Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

```
>>> top_dog = Restaurant(4)
>>> thai_basil = Restaurant(8)
>>> top_dog.serve_customer(mike)
>>> thai_basil.serve_customer(tajel)
>>> thai_basil.serve_customer(cecilia)
>>> len(top_dog.customer_history)
1
>>> len(thai_basil.customer_history)
2
```

1 +3.0
Correct

2 +0.0
Incorrect

What would be the most appropriate way to add `customer_history` to the Restaurant class?

Class attribute

Class method

Instance attribute

Instance method

Q5.3

4 points

I'd like to implement a `ChainRestaurant` class that behaves just like a `Restaurant`, but determines its `food_strength` by using the maximum `food_strength` from an input list of `Restaurant`s:

```
>>> chain_rest = ChainRestaurant(
  [Restaurant(5), Restaurant(6), Restaurant(2)]
)
>>> chain_rest.food_strength
6
>>> bob = Person("Bob")
>>> bob.hunger
10
>>> chain_rest.serve_customer(bob)
wow Bob is still hungry
>>> bob.hunger
4
```

I claim that we can implement this behavior by making a single change in the `ChainRestaurant` constructor. Fill in the blank to achieve the desired behavior:

```
class ChainRestaurant(Restaurant):
    def __init__(self, restaurants):
        super().__init__(_________FILL_ME_IN__________)
```

max([restaurant.food_strength for restaurant in restaurants])

⚠ Removing the **Correct** and **Incorrect** rubric items will inter
with auto-grading for this question.

1 +4.0

Correct: max([restaurant.food_strength for restaur
restaurants])

2 +4.0

Correct

3 +0.0

Incorrect

4 +3.5

Nearly Correct: minor mistake, like: passing `self` i
constructor. Ex: self.food_strength for i in restaurar

5 +2.0

Incorrect but on the right path. Ex: tries to do som
reasonable with restaurant.food_strength and taki
max. Ex: max(restaurants.food_strength), max for
[restaurants.value], max(food_strength for restaur
restaurants)

6 +0.0

Incorrect: not on the right path. Ex: max(food_stre
max(Restaurant)

+ Add Rubric Item | **Create Group** | **Ir**

Q5.4

4 points

Rubric

I'd like to implement a `MagicRestaurant` class that inherits from `Restaurant`. `MagicRestaurant` is special in that, in its `serve_customer()` method, it sets the customer's `hunger` to exactly `0` (aka "serves a magical food that makes every customer exactly full"):

```
>>> magic_restaurant = MagicRestaraunt()
>>> alice = Person("Alice")
```

⚠ Removing the **Correct** and **Incorrect** rubric items will inter
with auto-grading for this question.

1 +4.0

Correct

2 +0.0

```
>>> alice.hunger
10
>>> magic_restaurant.serve_customer(alice)
Alice is full
>>> alice.hunger
0
>>> bob = Person("Bob")
>>> bob.hunger = 4
>>> magic_restaurant.serve_customer(bob)
Bob is full
>>> bob.hunger
0
```

Incorrect

+ Add Rubric Item | Create Group | Ir

Which of the following implementations correctly implements the above desired behavior?

```
# Choice A
class MagicRestaraunt(Restaurant):
    def serve_customer(self, person):
        person.hunger = 0
        return super().serve_customer(person)

# Choice B
class MagicRestaraunt(Restaurant):
    def __init__(self):
        super().__init__(person.hunger)

# Choice C
class MagicRestaraunt(Restaurant):
    def __init__(self):
        super().__init__(0)
    def serve_customer(self, person):
        out = super().serve_customer(person)
        person.hunger = 0
        return out

# Choice D
class MagicRestaraunt(Restaurant):
    def __init__(self):
        super().__init__("magic")
    def serve_customer(self, person):
        self.food_strength = person.hunger
        return super().serve_customer(person)
```

Choice A

Choice B

Choice C

Choice D

Q6 Linked Lists

12 points

In this problem, you are to implement a function similar to Python's built-in `filter` function, but for linked lists. You will create a recursive function that takes a predicate function (a function that returns `True` or `False`) and a linked list, and returns a new linked list containing only the elements that satisfy the predicate.

You are provided with a `Link` class that represents a node in a linked list. The `Link` class is defined as follows:

```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest
```

```
def linked_list_filter(func, lnk):
    """Filters the linked list based on the predicate function.

    Parameters
    func (function): A function that takes a single argument
        and returns a boolean.
    lnk (Link): A linked list.

    Returns: A new linked list containing only the elements that
        satisfy the predicate function.

>>> def is_even(x):
        return x % 2 == 0
>>> lst = Link(1, Link(2, Link(3, Link(4, Link(5)))))
>>> filtered_lst = linked_list_filter(is_even, lst)
>> filtered_lst
Link(2, Link(4))
    """
if ___(a)___ == Link.empty:
    return Link.empty
elif func(___(b)___):
    return Link(___(c)___, ___(d)___)
else:
    return ___(e)___
```

Q6.1

2 points

Rubric

Fill in blank (a).

- `lnk.rest`
- `lnk`
- `lnk.rest.first`
- `lnk.first`

⚠ Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1 **+2.0**

Correct

2 **+0.0**

Incorrect

+ Add Rubric Item | Create Group | Ir

Q6.2

3 points

Rubric S

Fill in blank (b)

lnk.first

Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1 +3.0

Correct

2 +0.0

Incorrect

+ Add Rubric Item | Create Group | Ir

Q6.3

4 points

Rubric S

Fill in blank (c) and (d) using the options below.

```
lnk.first, linked_list_filter(func, lnk.rest)
```
```
func(lnk.first), linked_list_filter(func, lnk.rest)
```
```
lnk.first, linked_list_filter(func, lnk)
```
```
lnk.rest.first, linked_list_filter(func, lnk.rest)
```

Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1 +4.0

Correct

2 +0.0

Incorrect

+ Add Rubric Item | Create Group | Ir

Q6.4

3 points

Rubric S

Fill in blank (e).

Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1 +3.0

linked_list_filter(func, lnk.rest)

Correct

2 +0.0

Incorrect

+ Add Rubric Item | Create Group | Ir

Q7 Trees

12 points

Implement `constellation_tree`, which takes in a `Tree` instance and *mutates it* so that all values at depth `k` are changed to be the string `"star"`. You may assume that `k` is always less than or equal to the depth of the input tree.

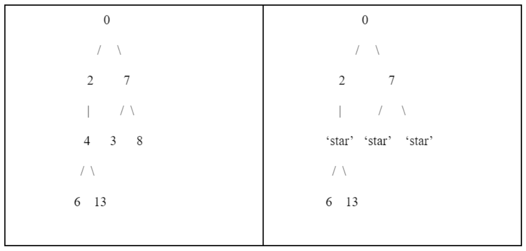
For reference, here is the `Tree` class definition:

```
class Tree:
    def __init__(self, value, branches=()):
        self.value = value
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)

    def is_leaf(self):
        return not self.branches
```

```
>>> t = Tree(
  0, [Tree(2, [Tree(4, [Tree(6), Tree(13)])]), Tree(7, [Tree(3), Tree(8)])]
)

>>> constellation_tree(t, 2)
>>> t
Tree(0, [Tree(2, [Tree('star', [Tree(6), Tree(13)])]), Tree(7, [Tree('star'), Tree('star')])])
```



```
def constellation_tree(t, k):
        if __________(a)__________:
                t.value = 'star'
        for b in ________(b)_________:
                constellation_tree(_____(c)_______)
```

Q7.1 Trees

4 points

Rubric

Fill in blank (a).

k == 0

Removing the **Correct** and **Incorrect** rubric items will inter
with auto-grading for this question.

1 +4.0

`k == 0`

2 -0.5

= instead of ==

3 +0.0

Incorrect

+ Add Rubric Item | **Create Group** | **Ir**

Q7.2

4 points

Rubric

Fill in blank (b).

t.branches

Removing the **Correct** and **Incorrect** rubric items will inter
with auto-grading for this question.

1 +4.0

`t.branches`

2 +0.0

Incorrect

+ Add Rubric Item | **Create Group** | **Ir**

Q7.3

4 points

Rubric

Fill in blank (c).

- b, k + 1
- b, k
- t, k - 1
- b, k - 1

Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1 +4.0
Correct

2 +0.0
Incorrect

+ Add Rubric Item | Create Group | Ir

Q8 Efficiency

11 points

Q8.1

3 points

Rubric

Recall: Both list indexing (eg `lst[ind]`) and `len(lst)` is a constant time O(1) operation.

```
def fn_a(lst):
    out = 0
    for ind in range(round(len(lst) / 2)):
        out += lst[ind]
    return out
```

What is the order of growth for `fn_a()`? Let `n` be the length of `lst`.

- O(1)
- O(log(n))
- O(n)
- O(n^2)
- O(2^n)

Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1 +3.0
Correct

2 +0.0
Incorrect

+ Add Rubric Item | Create Group | Ir

Q8.2

3 points

Rubric

Suppose the function `my_fn(lst)` takes in a list of integers and returns an integer, and is

Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

known to have order of growth O(n).

```
def fn_b(lst):
    out = 0
    for x in range(1000):
        out += my_fn(lst) * my_fn(lst)
    return out
```

What is the order of growth for `fn_b()`? Let n be the length of `lst`.

O(1)

O(log(n))

O(n)

O(n^2)

O(2^n)

1 +3.0

Correct

2 +0.0

Incorrect

Add Rubric Item | Create Group | Ir

Q8.3

3 points

Rubric

Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

Suppose `fn_c(n)` takes in an integer `n`.

```
def fn_c(n):
    for i in range(n):
        print('once')
    for j in range(n):
        print('twice')
```

What is the order of growth for `fn_c()`?

O(1)

O(log(n))

O(n)

O(n^2)

O(2^n)

1 +3.0

Correct

2 +0.0

Incorrect

Add Rubric Item | Create Group | Ir

Q8.4

2 points

Rubric

Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

Suppose `fn_d(n)` takes in an integer `n`.

```
def fn_d(n):
    if n <= 0:
```

1 +2.0

```
            print('zero!')
            return 0
        else:
            return fn_d(n - 1) + fn_d(n - 2)
```

Correct

2 +0.0

Incorrect

What is the order of growth for `fn_d()`?

O(1)

O(log(n))

O(n)

O(n^2)

O(2^n)

Add Rubric Item | Create Group | Ir

Q9 SQL

12 points

Suppose we have the following tables:

```
# Table: records
# name, department, title, salary, supervisor
Alyssa P Hacker, Computer, Programmer, 40000, Ben Bitdiddle
Ben Bitdiddle, Computer, Wizard, 60000, Oliver Warbucks
Eben Scrooge, Accounting, Chief Accountant, 75000, Oliver Warbucks
Lana Lambda, Administration, Executive Director, 610000, Lana Lambda
Lem E Tweakit, Computer, Technician, 25000, Ben Bitdiddle
Louis Reasoner, Computer, Programmer Trainee, 30000, Alyssa P Hacker
Oliver Warbucks, Administration, Big Wheel, 150000, Oliver Warbucks

# Table: salaries
# name, salary2022, salary2023
Alyssa P Hacker, 40000, 80000
Ben Bitdiddle, 60000, 80000
Eben Scrooge, 75000, 76000
Lana Lambda, 610000, 610000
Lem E Tweakit, 25000, 28000
Louis Reasoner, 30000, 30000
Oliver Warbucks, 150000, 120000

# Table: happy_table
# name, happiness_pts
Alyssa P Hacker, 8
Ben Bitdiddle, 6
Eben Scrooge, 2
Lana Lambda, 10
Lem E Tweakit, 6
Louis Reasoner, 6
Oliver Warbucks, 8
```

Q9.1

3 points

Rubric

I'd like to write a SQL query to fetch the name, salary, and title of all employees whose salary is > 70000:

```
# query template
SELECT ____________ FROM records WHERE ____________

# the desired query output
# name, salary, title
Eben Scrooge, 75000, Chief Accountant
Lana Lambda, 610000, Executive Director
Oliver Warbucks, 150000, Big Wheel
```

Note: the order of the result rows does not matter.

Write the correct SQL query, starting with the above query template:

SELECT name, salary, title FROM records WHERE salary > 70000;

⚠ Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1 +3.0
Correct: SELECT name, salary, title from records Wl salary > 70000;. Also correct: SELECT [name], [salar [title] ...

2 +3.0
Correct

3 +0.0
Incorrect

4 +2.5
Very Close: everything is right but didn't fetch all o right columns

5 +1.5
Sort of on the right path: wrote (mostly) valid SQL t kind of there but is incorrect

6 +0.0
Incorrect: shows no familiarity with SQL, or tries to Python code instead of SQL

7 -0.25
(subtractive) Minor SQL syntax issue or typo. Ex: SE [name, salary, title], or `>=` instead of `>`, 7000 vs 70

+ Add Rubric Item | Create Group | Ir

Q9.2

3 points

Rubric

I'd like to write a SQL query to calculate, for each supervisor, the maximum salary of the supervisor's subordinates (along with the supervisor's name):

⚠ Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

1 +3.0

```
SELECT ________ FROM records GROUP BY __________;

# the desired query output
Ben Bitdiddle, 40000
Oliver Warbucks, 150000
Lana Lambda, 610000
Alyssa P Hacker, 30000
Eben Scrooge, 18000
```

For instance, in the `records` table the supervisor `Ben Bitdiddle` has two subordinates: `Alyssa P Hacker` and `Lem E Tweakit`, with salaries `40000` and `25000` respectively. Hence, why we have the output row `Ben Bitdiddle, 40000`.

Note: the order of the result rows does not matter.

Write the correct SQL query, starting with the above query template:

SELECT supervisor, max(Salary) FROM records GROUP BY supervisor;

Correct: SELECT supervisor, max(Salary) FROM records GROU
supervisor;

2 +3.0

Correct

3 +2.5

Nearly Correct: everything is right, but they selecte `name` instead of `supervisor`, or selected unnecess columns. Ex: select name, max(salary) from record group by supervisor, select name, supervisor, max(from records group by supervisor;

4 +1.5

Has the right idea: query has the right idea (eg wa group on supervisor and apply max to salary), but query isn't right. Does the group but no max aggre Ex: select name,salary from records group by supe where map(max, salary), select supervisor,salary fr records group by supervisor.

5 +1.0

Almost has the right idea: the group is wrong, but a max on salary somewhere. Ex: select supervisor f records group by salary.max, select supervisor fro records group by max(salary)

6 +0.0

Incorrect

7 -0.25

(subtractive) Minor SQL syntax issue. Ex: SELECT [n salary].

+ Add Rubric Item | Create Group | Ir

Q9.3

3 points

Rubric S

⚠ Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

I'd like to write a SQL query to fetch both the old salary (from the `records` table) and the 2023 salary for each employee.

Here is a SQL query that tries to achieve this:

1 +3.0

```
SELECT name, salary, salary2023 FROM records, salaries
WHERE name=salaries.name;

# desired output
# name, salary, salary2023
Alyssa P Hacker, 40000, 80000
Ben Bitdiddle, 60000, 80000
Eben Scrooge, 75000, 76000
Lana Lambda, 610000, 610000
Lem E Tweakit, 25000, 28000
Louis Reasoner, 30000, 30000
Oliver Warbucks, 150000, 120000
```

Correct

2 +0.0

Incorrect

Add Rubric Item | Create Group | Ir

Note: the order of the result rows does not matter.

This query:

- Runs successfully and returns the desired output
- Runs successfully but returns the wrong output
- Errors

Q9.4

3 points

Rubric

Removing the **Correct** and **Incorrect** rubric items will inter with auto-grading for this question.

I'd like to write a SQL query that joins the records and happiness tables together to output the `name`, `salary`, and their `happiness_pts` together.

Here is a SQL query that tries to achieve this:

```
SELECT records.name, records.salary, happy_table.happine
FROM records, happy_table
WHERE records.name = happy_table.name;

# Desired output
# name, salary, happiness_pts
Alyssa P Hacker, 40000, 8
Ben Bitdiddle, 60000, 6
Eben Scrooge, 75000, 2
Lana Lambda, 610000, 10
Lem E Tweakit, 25000, 6
Louis Reasoner, 30000, 6
Oliver Warbucks, 150000, 8
```

1 +3.0

Correct

2 +0.0

Incorrect

Add Rubric Item | Create Group | Ir

Note: the order of the result rows does not matter.

This query:

- Runs successfully and returns the desired output
- Runs successfully but returns the wrong output
- Errors