

Your Name: _____

Your Student ID: _____

Exam Room: _____

Student ID of the person to your left: _____

Student ID of the person to your right: _____

You have 120 minutes. There are 9 questions of varying credit. (70 points total)

Ques-tion:	1	2	3	4	5	6	7	8	9	Total
Points:	12	8	10	9	8	6	9	8	0	70

For questions with **circular bubbles**, you may select only one choice.

- Unselected option (Completely unfilled)
- Don't do this (it will be graded as incorrect)
- Only one selected option (completely filled)

For questions with **square check boxes**, you may select one or more choices.

- You can select
- multiple squares
- (Don't do this)

Anything you write outside the answer boxes or you ~~cross out~~ will not be graded. If you write multiple answers, your answer is ambiguous, or the bubble/checkbox is not entirely filled in, we will grade the worst interpretation. For coding questions with blanks, you may write at most one statement per blank and you may not use more blanks than provided.

As a member of the UC Berkeley community, I act with honesty, integrity, and respect for others. I will follow the rules of this exam.

Acknowledge that you have read and agree to the honor code above and sign your name below:

Q1 Python is a Higher-Order Calling**(12 points)**

Assume that you have first started `python3` and executed the statements on the following box.

For each of the following parts, select or write what Python would display in the interactive interpreter. You can assume that all lines we've provided are executed. The output may have multiple lines. If an error occurs, write (or select) “**Error**”, but include all output displayed before the error. If evaluation would run forever, write “**Forever**”. If a function would be displayed, write “**Function**”.

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

```

1 from functools import reduce
2 add = lambda x, y: x + y
3 mul = lambda x, y: x * y
4 square = lambda x: x * x
5 is_odd = lambda x: x % 2 == 1 # True when x is odd.
6
7 nums = [ square(y) for y in range(1, 6) if is_odd(y) ]
8
9 lucky = (lambda x: lambda y: (y - x) % 8 == 0)(8)
10 def mystery(x, y):
11     if lucky(x):
12         print('ready')
13     while x > y:
14         x = x // 10
15         print(x)
16     return 'go'

```

Q1.1 (1 point) What would `python` display?

`>>> nums`

Q1.2 (1 point) What would `python` display?

`>>> reduce(square, range(1, 6))`

1

15

None of these

5

Error

Q1.3 (2 points) What would `python` display? Please indicate clearly where responses are on one or more lines.

`>>> mystery(40, 2)`

Q1.4 (2 points) What would `python` display? Please indicate clearly where responses are on one or more lines.

```
>>> print('c88c', mystery(7, 9))
```

Fill in the following four parts so that expression so that `list(second) == nums` will be True.

```
1 >>> first = ___(b)___ ( ___(a)___, range(1, 6))
2 >>> second = ___(d)___ ( ___(c)___, first)
3 >>> list(second) == nums
4 True
```

Q1.5 (1.5 points) Select the value that belongs in blank (a).

- | | | |
|----------------------------------------|-------------------------------------------|-------------------------------------|
| <input type="radio"/> <code>add</code> | <input type="radio"/> <code>square</code> | <input type="radio"/> None of these |
| <input type="radio"/> <code>mul</code> | <input type="radio"/> <code>is_odd</code> | |

Q1.6 (1.5 points) Select the value that belongs in blank (b).

- | | | | |
|----------------------------------------|-------------------------------------------|-------------------------------------------|-------------------------------------|
| <input type="radio"/> <code>map</code> | <input type="radio"/> <code>filter</code> | <input type="radio"/> <code>reduce</code> | <input type="radio"/> None of these |
|----------------------------------------|-------------------------------------------|-------------------------------------------|-------------------------------------|

Q1.7 (1.5 points) Select the value that belongs in blank (c).

- | | | |
|----------------------------------------|-------------------------------------------|-------------------------------------|
| <input type="radio"/> <code>add</code> | <input type="radio"/> <code>square</code> | <input type="radio"/> None of these |
| <input type="radio"/> <code>mul</code> | <input type="radio"/> <code>is_odd</code> | |

Q1.8 (1.5 points) Select the value that belongs in blank (d).

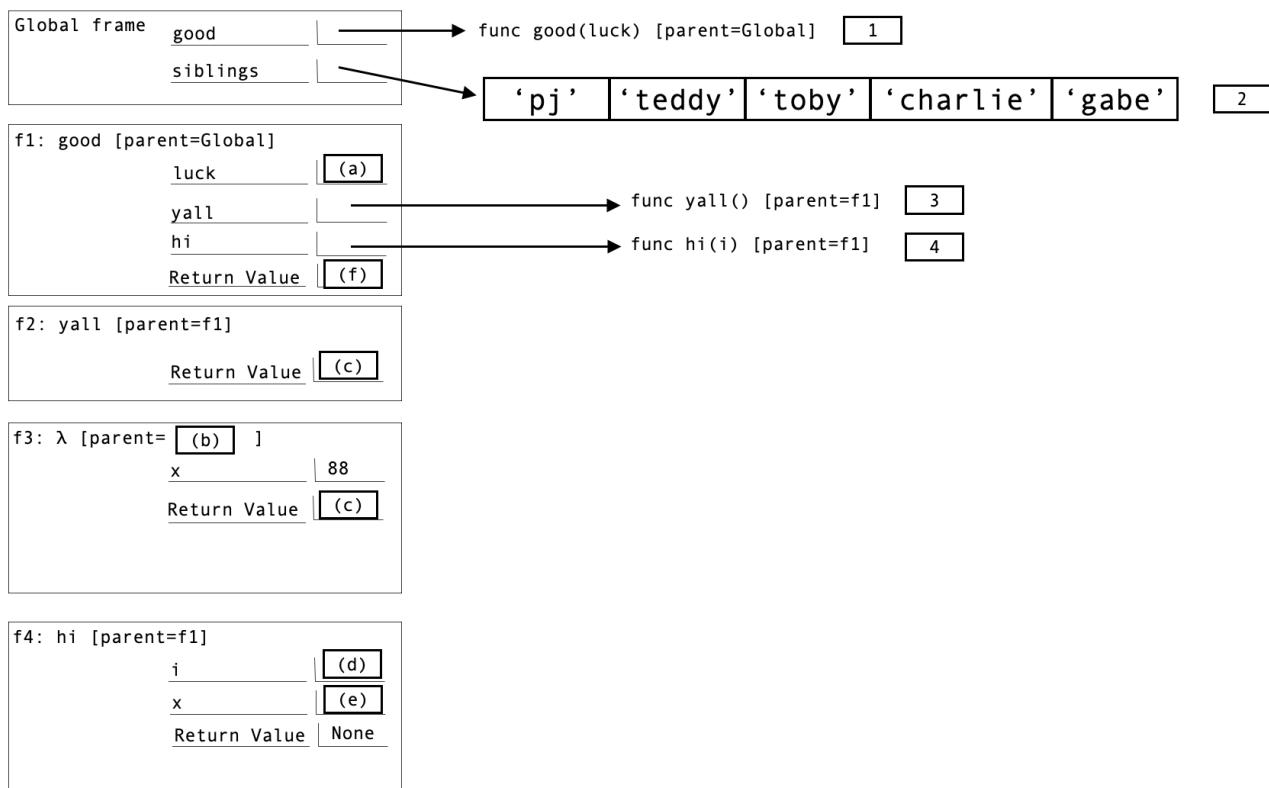
- | | | | |
|----------------------------------------|-------------------------------------------|-------------------------------------------|-------------------------------------|
| <input type="radio"/> <code>map</code> | <input type="radio"/> <code>filter</code> | <input type="radio"/> <code>reduce</code> | <input type="radio"/> None of these |
|----------------------------------------|-------------------------------------------|-------------------------------------------|-------------------------------------|

Q1.9 **Bonus Question – Do this last!** It will be worth up to 1 point of extra credit.

```
1 >>> is_it_fun = [ (lambda: square(x)) for x in range(1, 6) ]
2 >>> list(map(lambda f: f(), is_it_fun))
```

Q2 Good Luck 88**(8 points)**

Dhruv wants to watch TV, but he can only do so after completing the environment diagram of his favorite Disney show. Fill in the blanks to complete the environment diagram. Assume code has been fully run before filling in blanks. If there is a blank box with no label, there is no question associated with it and it is not scored. Please note: If more than one blank shares the same label (e.g., (d)), they have the same answer.



```

1 def good(luck):
2     def yall():
3         return (lambda x : len(luck) // 2)(88)
4     def hi(i):
5         for x in range(len(luck)):
6             if (i == len(luck)):
7                 i = 0
8                 print(luck[i])
9                 i += 1
10    return hi(yall())
11
12 siblings = ['pj', 'teddy', 'toby', 'charlie', 'gabe']
13 good(siblings)

```

Q2.1 (1 point) To what object/function/list does (a) point to? (The answer options refer to the numbered boxes in the environment diagram, not Python lists.)

 [1] [2] [3] [4]

Q2.2 (1 point) Fill in blank (b).

 Global f1 f2 f3

Q2.3 (1 point) Fill in blank (c).

 2 5 88 func yell() [parent = f1]

Q2.4 (1 point) After the last successful iteration, what is the value of (d)?

 1 2 3 4 5

Q2.5 (1 point) After the last successful iteration, what is the value of (e)?

 1 2 3 4 5

Q2.6 (1 point) Fill in blank (f).

 ["pj", "teddy", "toby", "charlie", "gabe"] ["toby", "charlie", "gabe", "pj", "teddy"] ["charlie", "gabe", "pj", "teddy", "toby"] ["gabe", "charlie", "toby", "teddy", "pj"] 2 None

Q2.7 (2 points) What is printed out to the console after line 13 is executed?

pj
teddy
toby
charlie
gabe

charlie
gabe
pj
teddy
toby

toby
charlie
gabe
pj
teddy

gabe
charlie
toby
teddy
pj

Q3 The Gauntlet of the Debug**(10 points)**

Sir Samuel Sweet is cleaning up the Berkeley Botanical Gardens and he needs your help to implement a function `cleanup_time` to calculate how much time he spends cleaning. He needs to avoid touching the flowers, needs to pull out the weeds, and needs to trim the bushes. The `garden` is represented by a list where each entry is a number that represents either a flower, weed, or bush.

- A number **divisible by 3 and 4 represents a flower**, a number **divisible by 3 (and not 4) represents a weed**, and a number **divisible by 4 (and not 3) represents a bush**. A number divisible by none of these options represents an unknown object to Sir Samuel Sweet and should be ignored.
- Each task takes a certain amount of time: avoiding a flower takes 0 minutes, pulling a weed takes 2 minutes, and trimming a bush takes 5 minutes.
- Sir Samuel Sweet is also on a time crunch and can only spend a **maximum time minutes cleaning** today. If he **exceeds time** minutes of work, the function `cleanup_time` should return -1.

Here is a table with the information for your convenience:

Task	Cost (Time in Minutes)	Conditions for Task
Flower	0	Divisible by 3 and 4
Weed	2	Divisible by 3 (and not 4)
Bush	5	Divisible by 4 (and not 3)

Table 1: Cleaning Guide

Like all Berkeley students, Isabelle was very tired and wrote a function that **contains bugs** which returns the total amount of time that Sir Samuel Sweet spends cleaning up the garden. Read through the code and answer the following questions (see next page).

```

1 def cleanup_time(garden, time):
2     """
3         >>> cleanup_time([10], 20) # Buggy implementation returns 0
4         0
5         >>> cleanup_time([30, 20], 6) # Buggy implementation returns 7
6         -1
7         >>> cleanup_time([15, 36], 10) # Buggy implementation returns 9
8         2
9         """
10    total = 0
11    index = 0
12    while index < len(garden) and time > 0:
13        task = garden[index]
14        if time < 0:
15            return -1
16        if task % 3 == 0 and task % 4 == 0:
17            total += 0
18            time -= 0
19        if task % 3 == 0:
20            total += 2
21            time -= 2
22        if task % 4 == 0:
23            total += 5
24            time -= 5
25        index += 1
26    return total

```

Q3.1 (1 point) What does `cleanup_time([15], 0)` return with the **current buggy implementation** of the function?

1

6

0

15

Q3.2 (2 points) Select all options where `cleanup_time` returns what it **should, according to the problem description**.

`cleanup_time([67], 67)`

`cleanup_time([15, 36], 6)`

`cleanup_time([20, 15, 18], 8)`

`cleanup_time([33, 30], 10)`

Q3.3 (1 point) What is the return value of `cleanup_time([8, 20], 6)` in the **current buggy implementation** of the function?

Q3.4 (1 point) What **should** the return value of `cleanup_time([8, 20], 6)` be assuming a **correct** implementation of the function?

Q3.5 (1 point) Select the numbers that will **NOT** be calculated correctly in the **current buggy implementation** of the `cleanup_time` function, regardless of the value of `time`.

15

24

28

25

48

Q3.6 (4 points) In each box, explain 1 bug in the code in 1-2 sentences. There are 2 distinct bugs, 1 for each box. Please include line numbers where the bug occurs, and note that **a single bug may be present on multiple lines**.

Q4 Even Page Rage**(9 points)**

Alicia and Maryam are participating in a reading challenge and want to read `pages_goal` pages total, but their pet peeve is that **they can only read books with an even number of pages**. Their favorite librarian gives them a set of popular series they might enjoy, `book_dict`, where each key is the name of a series, and each value is a list of the page counts of each book in the series.

After going through all the series the librarian has provided in `book_dict`, if they do not reach their goal of reading `pages_goal` pages, the function should print "try again tomorrow!". If they do reach their goal, the function should print "well done!". Additionally, the function should **return a list of the page counts of the books they were able to read**. (The order of the page counts in the returned list does not matter.)

```

1 def even_page_rage(pages_goal, book_dict):
2     """
3     >>> pages_goal = 1000
4
5     >>> oakland_library_dict = {
6         ...     "The Giver": [226, 241, 192, 336], # 226 + 192 + 336
7         ...     "The Hunger Games": [374, 391, 390] # 374 + 390
8         ... } # 226 + 192 + 336 + 374 + 390 = 1518
9     >>> result1 = even_page_rage(pages_goal, oakland_library_dict)
10    well done!
11    >>> sorted(result1) == [192, 226, 336, 374, 390]
12    True
13
14    >>> berkeley_library_dict = {
15        ... "Harry Potter": [223, 251, 317, 636, 767, 607, 607], # 636
16        ... "Percy Jackson": [377, 279, 312, 361, 381] # 312
17        ... } # 636 + 312 = 948
18    >>> result2 = even_page_rage(pages_goal, berkeley_library_dict)
19    try again tomorrow!
20    >>> sorted(result2) == [312, 636]
21    True
22    """
23
24    total_pages_read = ___(a)___
25    pages_read = []
26    for series in ___(b)___:
27        for page_count in ___(c)___:
28            if ___(d)___:
29                total_pages_read += ___(e)___
29                pages_read.append(___ (f)___)
30            if ___(g)___:
31                print('try again tomorrow!')
32            else:
33                print(___ (h)___)
34    return pages_read

```

Q4.1 (1 point) Fill in blank (a).

Q4.2 (2 points) Fill in blank (b).

Q4.3 (1 point) Fill in blank (c).

Q4.4 (1 point) Fill in blank (d).

Q4.5 (1 point) Fill in blank (e).

Q4.6 (1 point) Fill in blank (f).

Q4.7 (1 point) Fill in blank (g).

Q4.8 (1 point) Fill in blank (h).

Q5 Too Many Matcha Shops in Berkeley**(8 points)**

Grace B. and Reema want to go on a matcha run after class. They will only go to stores that (1) sell a specific drink, and (2) are close enough to Dwinelle.

You will implement the function `matcha_trip`, a three-layer higher-order function. The inputs and outputs of the 3 functions are described below.

1. `matcha_trip` takes in three arguments:

- `locations` (`dict`): maps store name to its position, represented as a list of its [`longitude`, `latitude`]
- `menus` (`dict`): maps store name to a list of drink names
- `dist_func` (`func`): takes 2 `ints`, `lon` and `lat`, and returns a `float` distance from Dwinelle
- `matcha_trip` returns `within` (`func`)
- It is safe to assume the keys of `locations` and `menus` will be the same

2. `within` takes in:

- `max_miles` (`int`): maximum distance in miles Grace and Reema are willing to travel and returns `find_drink` (`func`).

3. `find_drink` takes in:

- `drink_name` (`str`): the drink they are searching for and returns a list of store names that (1) sell `drink_name`, and (2) are **strictly within** `max_miles` according to the `dist_func`.

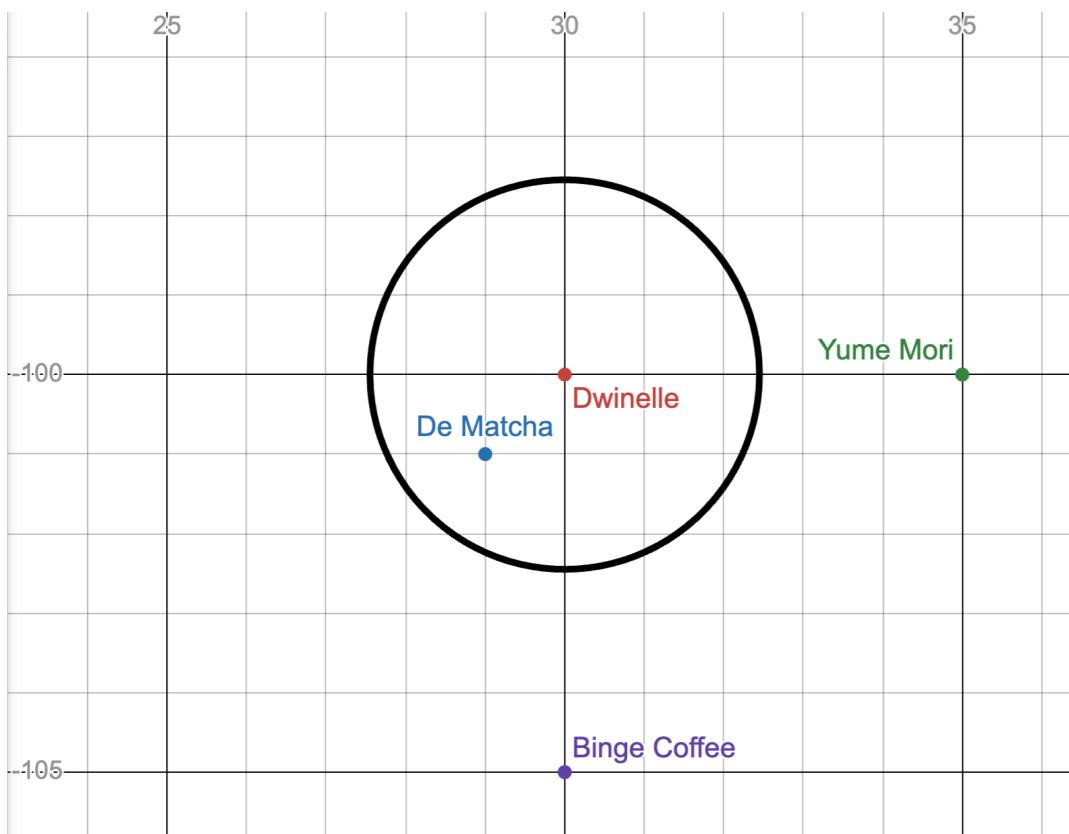
Here are the doctests for this problem (see next page for a diagram and the skeleton code):

```

1 >>> matcha_locations = {
2 ...     "De Matcha": [29, -101],
3 ...     "Yume Mori": [35, -100],
4 ...     "Binge Coffee" : [30, -105]
5 ...
6 >>> drink_menus = {
7 ...     "De Matcha": ["Strawberry Matcha", "Mango Matcha"],
8 ...     "Yume Mori": ["Double Matcha", "Matcha Tiramisu"],
9 ...     "Binge Coffee": ["Strawberry Matcha", "Matcha Latte"]
10 ...
11 >>> dwinelle = [30, -100]
12 >>> dist_func = # Assume this returns the Euclidean distance from Dwinelle
13 >>> within_dist = matcha_trip(matcha_locations, drink_menus, dist_func)
14 >>> find_drink = within_dist(3)
15 >>> find_drink("Strawberry Matcha")
16 ['De Matcha']
17 >>> find_drink("Matcha Latte")
18 []
19 >>> find_drink("Blackberry Espresso Tonic")
20 []

```

Notice in the diagram below that De Matcha is located within 3 miles of Dwinelle since it is inside the circle with radius 3 (whereas Yume Mori and Binge Coffee are not). Thus, when searching for "Strawberry Matcha", only De Matcha is returned.



```

1 def matcha_trip(locations, menus, dist_func):
2     def within(max_miles):
3         def find_drink(drink_name):
4             stores = []
5             for store_name in locations:
6                 if ___(a)___ in ___(b)___:
7                     if dist_func(___ (c)___[0], ___(c)___[1]) < max_miles:
8                         ___(d)_____.extend(___ (e)___)
9             return stores
10        return ___(f)___
11    return ___(g)___

```

Q5.1 (1 point) Fill in blank (a).

Q5.2 (1 point) Fill in blank (b).

Q5.3 (1 point) Fill in blank (c).

Q5.4 (1 point) Fill in blank (d).

Q5.5 (2 points) Fill in blank (e).

Q5.6 (1 point) Fill in blank (f).

Q5.7 (1 point) Fill in blank (g).

Q6 Book of Blahaj**(6 points)**

You work on IKEA's marketing team and have different furniture items that you'd like to advertise to customers in different magazines. Let's do this with ADTs!

Below are 2 ADTs, Furniture and Magazine:

1. Furniture ADTs have 3 attributes: `name` (string), `price` (integer), and `size` (string: '`Small`', '`Medium`', or '`Large`').
2. Magazine ADTs have 2 attributes: `title` (string) and `pages` (list of Furniture ADTs in that magazine).

We have also defined some Furniture ADTs: `blahaj`, `alex`, `kallax`, and `hauga`, which may be used throughout the doctests in the subparts below.

```

1 # Furniture ADT
2 def make_furniture(name, price, size):
3     return [name, price, size]
4
5 def get_name(furniture):
6     return furniture[0]
7
8 def get_price(furniture):
9     return furniture[1]
10
11 def get_size(furniture):
12     return furniture[2]
13
14 # Magazine ADT
15 def make_magazine(title, pages): # pages is a list of Furniture ADTs
16     return {'title': title, 'pages': pages}
17
18 def get_title(magazine):
19     return magazine['title']
20
21 # Returns list of furniture ADTs
22 def get_pages(magazine):
23     return magazine['pages']
24
25 # Initialize some furniture to use in the doctests below
26 blahaj = make_furniture('Blahaj Toy', 30, 'Small')
27 alex = make_furniture('Alex Drawer', 50, 'Medium')
28 kallax = make_furniture('Kallax Shelf', 80, 'Large')
29 hauga = make_furniture('Hauga Table', 100, 'Large')
```

Q6.1 - Q6.2: Consider the following `discount` function, which does **NOT** belong to any ADT. It takes in a furniture ADT and `percent` off (as a float, e.g. 0.1 means 10% off) and returns the discounted price of the furniture.

```

1 def discount(furniture, percent):
2     """
3     >>> discount(hauga, 0.1)
4     90.0
5     """
6     return furniture[1] * (1 - percent)

```

Q6.1 (1 point) Does the `discount` function work as intended?

Yes

No

Q6.2 (1 point) Does the `discount` function violate any abstraction barriers?

Yes

No

Q6.3 - Q6.5: Implement the function `group_by_size` which takes in a magazine and returns a dictionary where the keys are the sizes and the values are lists of Furniture ADTs with the corresponding size. The order of furniture ADTs should be maintained: i.e. in the doctest, the list containing the large furniture items must be `[kallax, hauga]` NOT `[hauga, kallax]`.

```

1 def group_by_size(magazine):
2     """
3     >>> pages = [kallax, blahaj, alex, hauga]
4     >>> m = make_magazine('Ikea Magazine', pages)
5     >>> grouped = group_by_size(m)
6     >>> grouped['Small'] == [blahaj]
7     True
8     >>> grouped['Medium'] == [alex]
9     True
10    >>> grouped['Large'] == [kallax, hauga]
11    True
12    """
13    result = {'Small': [], 'Medium': [], 'Large': []}
14    for furniture in ____(a)____:
15        result[____(b)____].____(c)____
16    return result

```

Q6.3 (1 point) Fill in blank (a).

Q6.4 (1 point) Fill in blank (b).

Q6.5 (2 points) Which of the following could go in blank (c)? Select all that apply.

- `append(furniture)`
- `extend(furniture)`
- `insert(0, furniture)`

- `append([furniture])`
- `extend([furniture])`
- `pop(0, furniture)`

Q7 Demon Slayer Recursion**(9 points)**

In the world of *Demon Slayer*, Tanjiro relies on crows to deliver battle reports. The problem is, these crows are very dramatic. Instead of speaking normally, the crows stretch out their words by screaming the same syllable again and again. So a simple message like "saveme" might arrive as "saaavemeeee".

Tanjiro doesn't have time to write down every single repeated scream. Instead, he wants to compress the crow's messages so that long runs of the same sound are written as "letter + count".

For example:

- "saaavemeeee" should become "s1a3v1e1m1e4"
- "caaaaaw" should become "c1a5w1"

However, a completely silent crow message cannot be compressed, so **you may assume the input string consists only of lowercase letters and is never empty**. Help Tanjiro by implementing the recursive function `compress`.

```

1 def compress(s):
2     """
3         >>> compress("demon")
4         'd1e1m1o1n1'
5         >>> compress("booya")
6         'b1o2y1a1'
7         >>> compress("aaaa")
8         'a4'
9         >>> compress("saaavemeeee")
10        's1a3v1e1m1e4'
11        """
12
13     def helper(s, current, count):
14         partial = current + f"{{}_{(a)}_{}}"
15         if s == "":
16             return partial
17         if ___(b)___ == current:
18             return helper(s[1:], current, ___(c)___)
19         else:
20             return ___(d)___ + ___(e)___
21     return ___(f)___

```

Q7.1 (1 point) Choose the correct choice to fill in blank (a). Recall that an f-string in Python allows you to format strings like so:

```

>>> name = "Bob"
>>> age = 18
>>> print(f"I'm {name} and I'm {age} years old")
I'm Bob and I'm 18 years old

```

 s current count count + 1

Q7.2 (1 point) Fill in blank (b).

Q7.3 (2 points) Fill in blank (c).

Q7.4 (2 points) Fill in blank (d).

Q7.5 (1 point) Choose the correct choice to fill in blank (e)

- helper(s, s[0], 0)
- helper(s, s[0], 1)
- helper(s[1:], s[0], 0)
- helper(s[1:], s[0], 1)
- helper(s[1:], s[1], 1)

Q7.6 (2 points) Fill in blank (f).

Q8 Packing Dilemma**(8 points)**

You want to bring your nicest jewelry to vacation, however your suitcase has a weight limit. Luckily, you can use your Python skills to solve this problem! Implement `suitcase`, which takes in:

- A list of positive integers `weights`
- A list of positive integers `values`
- A non-negative weight capacity `p`

It returns the **max value** of your jewelry that fits within the capacity (you want to pick some subset of the items so that you maximize the value you're bringing). Assume that the item at index `i` weighs `weights[i]` pounds, and is worth `values[i]` dollars. You may also assume that the lengths of `weights` and `values` are always the same.

```

1 def suitcase(weights, values, p):
2     """
3     >>> suitcase([], [], 5)
4     0
5     >>> suitcase([2, 3], [5, 6], 0)
6     0
7
8     >>> weights = [1, 3, 4]
9     >>> values = [3, 10, 12]
10
11    >>> suitcase(weights, values, 4) # pick items at indexes 0 and 1
12    13
13    >>> suitcase(weights, values, 6) # pick items at indexes 0 and 2
14    15
15    """
16
17    if ___(a)___:
18        return 0
19    else:
20        first_weight = weights[0]
21        rest_weights = weights[1:]
22        first_value = values[0]
23        rest_values = values[1:]
24
25        with_first = first_value + ___(b)___
26        without_first = ___(c)___
27
28        if first_weight <= p:
29            return ___(d)___
30        else:
31            return without_first

```

Q8.1 (2 points) Select all the correct choices for blank (a).

- `weights == [[]] and p == 0`
- `weights == [] and p > 0`
- `weights == [] or p == 0`
- `len(weights) == 0 or p == 0`

Q8.2 (2 points) Fill in blank (b).

Q8.3 (2 points) Fill in blank (c).

Q8.4 (2 points) Fill in blank (d).

Q9 Just for fun!**(0 points)**

Q9.1 Draw something fun, or write a message for the staff!

This page left intentionally (mostly) blank

Please do not tear off any pages from the exam.