DATA C88C
Fall 2025

Michael Ball

Midterm

Solutions last updated: Tuesday, October 28, 2025

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)

Question:	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
```

```
[1, 9, 25]
```

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)
```

ready
4
0
'go'

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

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

c88c go

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

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

Solution: Note that there were 2 valid solutions for Q1.5 - Q1.8 and during this semester we accepted either. We graded all 4 subparts together and gave partial credit accordingly.

Alternate solution:

(a) `square`

(b) `map`

(c) `is_odd`

(d) `filter`

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

- ○ `add`
- ○ `square`
- ○ None of these
- ○ `mul`
- ● `is_odd`

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

- ○ `map`
- ● `filter`
- ○ `reduce`
- ○ None of these

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

- ○ `add`
- ● `square`
- ○ None of these
- ○ `mul`
- ○ `is_odd`

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

- ● `map`
- ○ `filter`
- ○ `reduce`
- ○ None of these

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

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

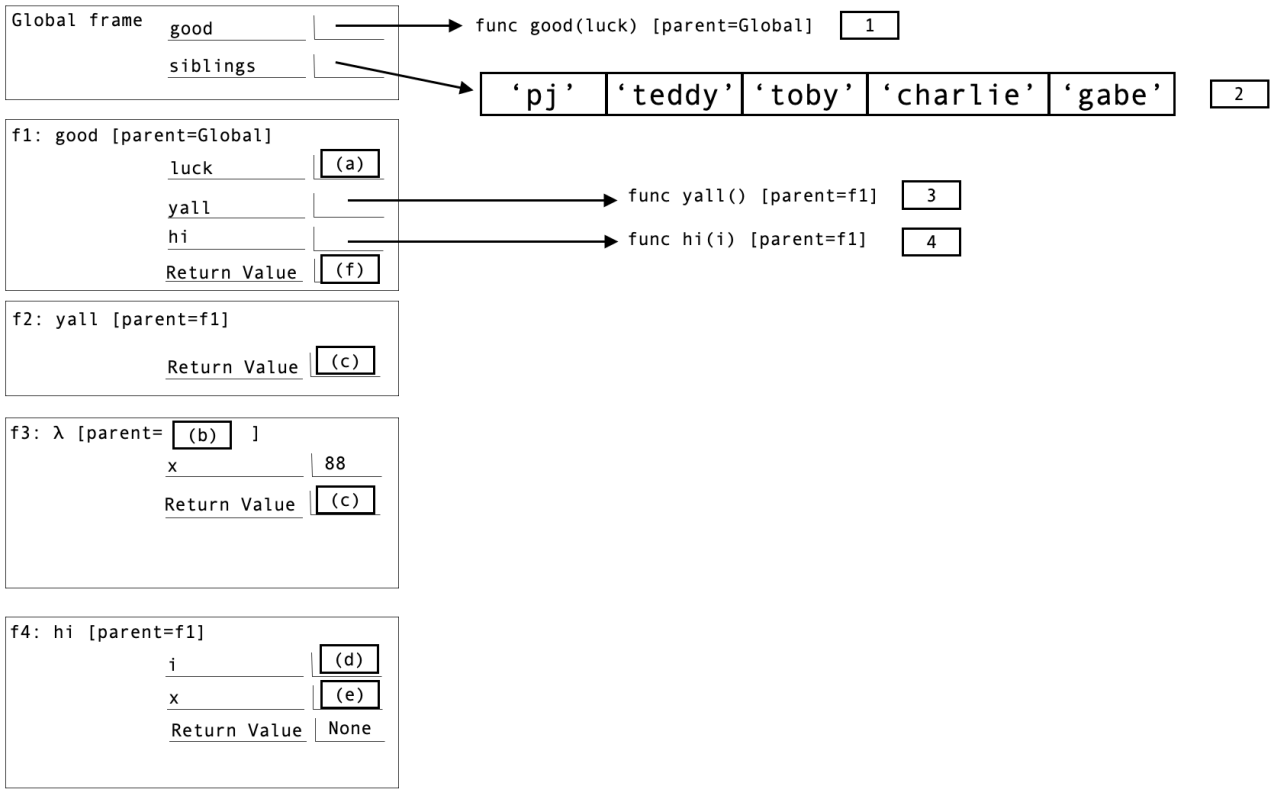
```
[25, 25, 25, 25, 25]
```

SID: _____

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 yall() [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).

```
def cleanup_time(garden, time):
  """
  >>> cleanup_time([10], 20) # Buggy implementation returns 0
  0
  >>> cleanup_time([30, 20], 6) # Buggy implementation returns 7
  -1
  >>> cleanup_time([15, 36], 10) # Buggy implementation returns 9
  2
  """
  total = 0
  index = 0
  while index < len(garden) and time > 0:
    task = garden[index]
    if time < 0:
      return -1
    if task % 3 == 0 and task % 4 == 0:
      total += 0
      time -= 0
    if task % 3 == 0:
      total += 2
      time -= 2
    if task % 4 == 0:
      total += 5
      time -= 5
    index += 1
  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?

10

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

−1

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**.

The use of `if` statements accidentally triple counts when we have an element divisible by 3 and 4. We should use `if`, `elif` statements to ensure that we don't count tasks more than once. [Lines: 19, 22]

Intended solution: `-1` is not being returned when Sir Samuel Sweet works overtime since `time` may be decremented after the less-than-zero check in the `while` loop of the buggy implementation. We should perform the `time < 0` check (which determines whether to `return -1`) after the `while` loop terminates or after the `time` variable is being updated, rather than at the start of each iteration [Lines: 14, 15]. Otherwise, we might incorrectly return a positive `total` even though `time` has already run out.

Alternate solution we are accepting: Time condition `time > 0` in while loop should be deleted to allow us to check the condition `time < 0`. [Line 12]

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.)

```
def even_page_rage(pages_goal, book_dict):
  """
  >>> pages_goal = 1000

  >>> oakland_library_dict = {
  ...    "The Giver": [226, 241, 192, 336],  # 226 + 192 + 336
  ...    "The Hunger Games": [374, 391, 390]  # 374 + 390
  ... } # 226 + 192 + 336 + 374 + 390 = 1518
  >>> result1 = even_page_rage(pages_goal, oakland_library_dict)
  well done!
  >>> sorted(result1) == [192, 226, 336, 374, 390]
  True

  >>> berkeley_library_dict = {
  ...  "Harry Potter": [223, 251, 317, 636, 767, 607, 607], # 636
  ...  "Percy Jackson": [377, 279, 312, 361, 381] # 312
  ... } # 636 + 312 = 948
  >>> result2 = even_page_rage(pages_goal, berkeley_library_dict)
  try again tomorrow!
  >>> sorted(result2) == [312, 636]
  True
  """
  total_pages_read = ___(a)___
  pages_read = []
  for series in ___(b)___:
    for page_count in ___(c)___:
      if ___(d)___:
        total_pages_read += ___(e)___
        pages_read.append(___(f)___)
  if ___(g)___:
    print('try again tomorrow!')
  else:
    print(___(h)___)
  return pages_read
```

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

```
0
```

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

`book_dict.values()` or `book_dict`

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

If (b) was `book_dict.values()`, then (c) is `series`. Otherwise, if (b) was `book_dict`, then (c) is `book_dict[series]`

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

```
page_count % 2 == 0
```

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

```
page_count
```

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

```
page_count
```

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

`total_pages_read < pages_goal` or `sum(pages_read) < pages_goal`

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

```
'well done!'
```

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.

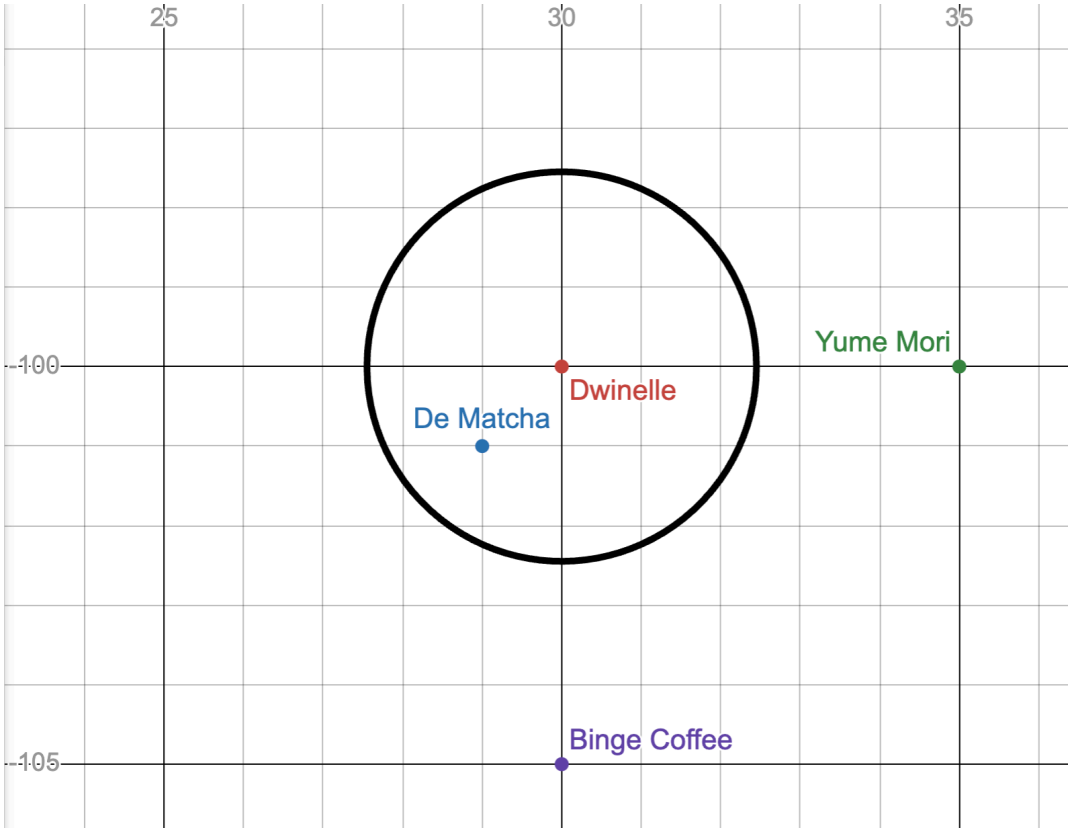
---

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):

```
>>> matcha_locations = {
...   "De Matcha": [29, -101],
...   "Yume Mori": [35, -100],
...   "Binge Coffee" : [30, -105]
... }
>>> drink_menus = {
...   "De Matcha": ["Strawberry Matcha", "Mango Matcha"],
...   "Yume Mori": ["Double Matcha", "Matcha Tiramisu"],
...   "Binge Coffee": ["Strawberry Matcha", "Matcha Latte"]
... }
>>> dwinelle = [30, -100]
>>> dist_func = # Assume this returns the Euclidean distance from Dwinelle
>>> within_dist = matcha_trip(matcha_locations, drink_menus, dist_func)
>>> find_drink = within_dist(3)
>>> find_drink("Strawberry Matcha")
['De Matcha']
>>> find_drink("Matcha Latte")
[]
>>> find_drink("Blackberry Espresso Tonic")
[]
```

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.




```
def matcha_trip(locations, menus, dist_func):
  def within(max_miles):
    def find_drink(drink_name):
      stores = []
      for store_name in locations:
        if ___(a)___ in ___(b)___:
          if dist_func(___(c)___[0], ___(c)___[1]) < max_miles:
            ___(d)___.extend(___(e)___)
      return stores
    return ___(f)___
  return ___(g)___
```

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

```
drink_name
```

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

```
menus[store_name]
```

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

```
locations[store_name]
```

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

```
stores
```

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

```
[store_name]
```

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

```
find_drink
```

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

```
within
```

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.

```
# Furniture ADT
def make_furniture(name, price, size):
  return [name, price, size]

def get_name(furniture):
  return furniture[0]

def get_price(furniture):
  return furniture[1]

def get_size(furniture):
  return furniture[2]

# Magazine ADT
def make_magazine(title, pages): # pages is a list of Furniture ADTs
  return {'title': title, 'pages': pages}

def get_title(magazine):
  return magazine['title']

# Returns list of furniture ADTs
def get_pages(magazine):
  return magazine['pages']

# Initialize some furniture to use in the doctests below
blahaj = make_furniture('Blahaj Toy', 30, 'Small')
alex = make_furniture('Alex Drawer', 50, 'Medium')
kallax = make_furniture('Kallax Shelf', 80, 'Large')
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.

```
def discount(furniture, percent):
    """
    >>> discount(hauga, 0.1)
    90.0
    """
    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

Solution: The `discount` function makes the assumption that the Furniture ADT is implemented as a list where the second element is the price, but it is possible that the ADT implementation changes over time. To fix the code, it should use `get_price` to retrieve the price.

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]`.

```
def group_by_size(magazine):
  """
  >>> pages = [kallax, blahaj, alex, hauga]
  >>> m = make_magazine('Ikea Magazine', pages)
  >>> grouped = group_by_size(m)
  >>> grouped['Small'] == [blahaj]
  True
  >>> grouped['Medium'] == [alex]
  True
  >>> grouped['Large'] == [kallax, hauga]
  True
  """
  result = {'Small': [], 'Medium': [], 'Large': []}
  for furniture in ___(a)___:
    result[___(b)___].___(c)___
  return result
```

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

```
get_pages(magazine)
```

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

```
get_size(furniture)
```

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

- [x] `append(furniture)`
- [ ] `append([furniture])`
- [ ] `extend(furniture)`
- [x] `extend([furniture])`
- [ ] `insert(0, 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     def helper(s, current, count):
13         partial = current + f"{___(a)___}"
14         if s == "":
15             return partial
16         if ___(b)___ == current:
17             return helper(s[1:], current, ___(c)___)
18         else:
19             return ___(d)___ + ___(e)___
20     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`

Solution: Alternatively, if you selected `count + 1` for (a) and `helper(s[1:], s[0], 0)` for blanks (e) and (f), that is also correct.

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

`s[0]`

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

`count + 1`

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

`partial`

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)`

Solution: Alternatively, if you selected `count + 1` for (a) and `helper(s[1:], s[0], 0)` for blanks (e) and (f), that is also correct.

`helper(s, s[0], 0)` was also accepted as a correct solution.

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

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

Solution: Alternatively, if you selected `count + 1` for (a) and `helper(s[1:], s[0], 0)` for blanks (e) and (f), that is also correct.

Solution: Note that there were many alternate answers besides the staff solution provided above. See the code below for the alternate solutions students had that we also accepted:

```
# Selected count + 1 for (a) AND
# Selected helper(s[1:], s[0], 0) for (e) AND
# Wrote helper(s[1:], s[0], 0) for (f)
# All other choices match the staff solution
def compress_alt_one(s):
    def helper(s, current, count):
        partial = current + f"{count + 1}" # (a)
        if s == "":
            return partial
        if s[0] == current:
            return helper(s[1:], current, count + 1)
        else:
            return partial + helper(s[1:], s[0], 0) # (e)
    return helper(s[1:], s[0], 0) # (f)
```

```
# Wrote helper(s, s[0], 0) for (f)
# All other choices match the staff solution
def compress_alt_two(s):
    def helper(s, current, count):
        partial = current + f"{count}"
        if s == "":
            return partial
        if s[0] == current:
            return helper(s[1:], current, count + 1)
        else:
            return partial + helper(s[1:], s[0], 1)
    return helper(s, s[0], 0) # (f)
```

```
# Selected helper(s, s[0], 0) for (e)
# All other choices match the staff solution
def compress_alt_three(s):
    def helper(s, current, count):
        partial = current + f"{count}"
        if s == "":
            return partial
        if s[0] == current:
            return helper(s[1:], current, count + 1)
        else:
            return partial + helper(s, s[0], 0) # (e)
    return helper(s[1:], s[0], 1)
```

Solution:

```
# Selected helper(s, s[0], 0) for (e) AND
# Wrote helper(s, s[0], 0) for (f)
# All other choices match the staff solution
def compress_alt_four(s):
    def helper(s, current, count):
        partial = current + f"{count}"
        if s == "":
            return partial
        if s[0] == current:
            return helper(s[1:], current, count + 1)
        else:
            return partial + helper(s, s[0], 0) # (e)
    return helper(s, s[0], 0) # (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.

```
def suitcase(weights, values, p):
  """
  >>> suitcase([], [], 5)
  0
  >>> suitcase([2, 3], [5, 6], 0)
  0

  >>> weights = [1, 3, 4]
  >>> values = [3, 10, 12]

  >>> suitcase(weights, values, 4) # pick items at indexes 0 and 1
  13
  >>> suitcase(weights, values, 6) # pick items at indexes 0 and 2
  15
  """
  if ___(a)___:
    return 0
  else:
    first_weight = weights[0]
    rest_weights = weights[1:]
    first_value = values[0]
    rest_values = values[1:]

    with_first = first_value + ___(b)___
    without_first = ___(c)___

    if first_weight <= p:
        return ___(d)___
    else:
        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`

Solution: The function is correct as-is, but we should technically check `p <= 0` instead of just `p == 0` to avoid making unnecessary recursive calls. However, since we only call `suitcase` with `p - first_weight` when `first_weight <= p`, it does not affect the correctness of the function.

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

```
suitcase(rest_weights, rest_values, p - first_weight)
```

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

```
suitcase(rest_weights, rest_values, p)
```

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

```
max(with_first, without_first)
```

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.