

INSTRUCTIONS

This is your exam. Complete it either at exam.cs61a.org or, if that doesn't work, by emailing course staff with your solutions before the exam deadline.

This exam is intended for the student with email address `<EMAILADDRESS>`. If this is not your email address, notify course staff immediately, as each exam is different. Do not distribute this exam PDF even after the exam ends, as some students may be taking the exam in a different time zone.

For questions with **circular bubbles**, you should select exactly *one* choice.

- You must choose either this option
- Or this one, but not both!

For questions with **square checkboxes**, you may select *multiple* choices.

- You could select this choice.
- You could select this one too!

You may start your exam now. Your exam is due at `<DEADLINE>` Pacific Time. Go to the next page to begin.

- **Online Exams:** You may start you exam as soon as you are given the password.
- **You may have a digital version of the CS88 Reference Sheet, or the PDF, but no other files.**

Exam Clarifications: https://docs.google.com/document/d/1-zKDRM84o6cr5lGel3C8CuF-ttPNkQVILAKa39_caDk/edit?usp=sharing

Reference Sheet: <https://drive.google.com/file/d/1vGqbiXnxiUu19hjrB9MNIzo5kd-al5iU/view>

1. (5.0 points) What Would Python Do (WWPD)

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 a function is returned, write “Function”.

```
>>> def f1(x, y):
    if x > (x + y):
        print(x)
        y = x
    if x > (x - y):
        print(y)
        x = y
    return x + y
>>> def f2(a, b):
    if a:
        return b and a
    else:
        return a or b
>>> f3 = lambda lst: lst[1:] + lst.pop(0)
```

(a) (1.0 pt) >>> f1(3, -5)

```
3
3
6
```

(b) (1.0 pt) >>> f1(-4, 7)

```
7
14
```

(c) (1.5 pt) >>> f2(print('a'), 10 % 3)

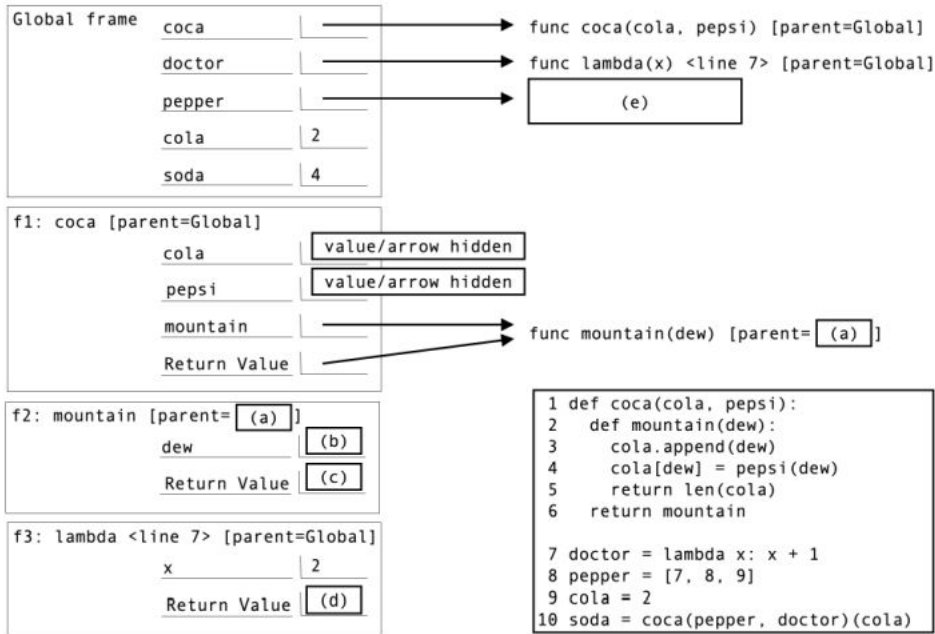
```
a
1
```

(d) (1.5 pt) >>> f3([[10, 20], 30, 40])

```
[30, 40, 10, 20]
```

2. (6.0 points) Soft Drinks

Fill in the blanks to complete the environment diagram. All the code used is in the box to the right, and the code runs to completion with no errors. Some arrows have been removed from the diagram. You may wish to draw in those arrows, but it is not required.



(a) (1.0 pt) What is the parent frame of the `mountain` function in the `f2` frame?

f1

(b) (1.0 pt) What is the value of `dew` in the `f2` frame?

2

(c) (1.0 pt) What is the return value of the `f2` frame?

4

(d) (1.0 pt) What is the return value of the `f3` frame?

3

(e) (2.0 pt) What is the value of `pepper` in the global frame when the environment diagram is complete?

[7, 8, 3, 2]

3. (6.0 points) BeReal

BeReal is a popular new social media app that sends alerts to users at a certain time in the day. Users then have 2 minutes after the alert to post a picture, otherwise their post will be marked as late.

Complete `bereal` that takes in an `alert` time and returns a function `capture`. `capture` takes in a `post` time and returns three possible strings:

- "on time" if the post is less than or equal to 2 minutes of the alert.
- "x minutes late" if the post is less than an hour late.
- "x hours late" if the post is greater than or equal to an hour late, where x is rounded down to the nearest hour.

The `alert` and `post` arguments are both integer values representing the time of day in minutes after midnight (e.g. 10 AM is 600 minutes), and the `post` time will never be earlier than the `alert` time. (`str(x)` converts a number x to a string.)

```
def bereal(alert):
    """
    >>> today = bereal(600) # 10:00 AM
    >>> today(601) # 10:01 AM
    'on time'
    >>> today(602) # 10:02 AM
    'on time'
    >>> today(630) # 10:30 AM
    '30 minutes late'
    >>> today(730) # 12:10 PM
    '2 hours late'
    """
    def capture(post):
        difference = _____
        if _____:
            return str(_____) + ' hours late'
        elif _____:
            return str(_____) + ' minutes late'
        else:
            return 'on time'
    _____
```

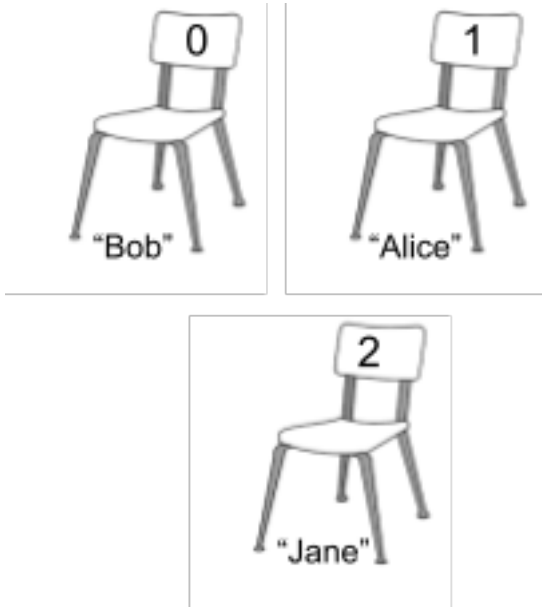
(a) (6.0 pt)

```
def bereal(alert):
    def capture(post):
        difference = post - alert
        if difference >= 60:
            return str(difference // 60) + ' hours late'
        elif difference > 2:
            return str(difference) + ' minutes late'
        else:
            return 'on time'
    return capture
```

4. (6.0 points) Musical Chairs

A group of people sitting in a circle of chairs can be represented as a list of strings. We'll call this representation a "chair list" and look at an example of a "chair list" named `people`.

```
>>> people = ["Bob", "Alice", "Jane"]
```



There are `len(people)` total chairs in the circle and they are labeled `0, 1, ..., len(people) - 1` in the clockwise direction. The i -th element in the list `people` is the name of the person in the chair labeled with integer i .

- (a) (2.0 pt) Complete the function `shift_right` which takes in a "chair list" named `people` and returns a new "chair list" in which every person moves to the chair directly to their right. The original list `people` should not be modified.

```
def shift_right(people):
    """
    >>> p0 = ["Bob", "Alice", "Jane"]
    >>> p1 = shift_right(p0)
    >>> p0
    ['Bob', 'Alice', 'Jane'] # p0 remains unchanged
    >>> p1
    ['Jane', 'Bob', 'Alice']
    >>> p2 = shift_right(p1)
    >>> p2
    ['Alice', 'Jane', 'Bob']
    """
    return -----
```

```
return [people[-1]] + people[:-1]
```

- (b) (4.0 pt) Complete the function `shift_right_n_times` which takes in a “chair list” named `people` and an integer `n` and returns a new “chair list” in which every person moves `n` chairs to their right. The original list `people` should not be modified.

```
def shift_right_n_times(people, n):  
    """  
    >>> shift_right_n_times(['Bob', 'Alice', 'Jane'], 2)  
    ['Alice', 'Jane', 'Bob']  
    >>> shift_right_n_times(['W', 'X', 'Y', 'Z'], 8)  
    ['W', 'X', 'Y', 'Z']  
    """
```

In your solution, you may assume that `shift_right` correctly. You may not need to use all the lines provided. (You may use any valid Python you have learned in CS88 in your solution.)

```
def shift_right_n_times(people, n):  
    for _ in range(n):  
        people = shift_right(people)  
    return people
```

5. (6.0 points) Aggregate

Complete the `aggregate` function, which aggregates certain digits of `n` from right to left using the two argument function `func`. `aggregate` only aggregates a digit `d` if `cond(d)` evaluates to `True` and `d` is not consecutively repeated. For example, if you have `n = 222` and the digit 2 satisfies `cond`, you will only aggregate using one of the three consecutive 2's. Finally, the first time `func` is called on a digit of `n`, pass `base` as one of the arguments. Note that the order in which arguments are passed to `func` does not matter, i.e. `func(x, y)` equals `func(y, x)`.

```
def aggregate(n, func, cond, base):
    """
    >>> from operator import add, mul
    >>> add_times_three = lambda x, y: (x + y) * 3
    >>> is_even = lambda x: x % 2 == 0
    >>> aggregate(122, add_times_three, is_even, 0) # (0 + 2) * 3
    6
    >>> aggregate(4212, add, is_even, 0) # (((0 + 2) + 2) + 4)
    8
    >>> aggregate(222, add, is_even, 1) # 1 + 2
    3
    >>> aggregate(111, mul, is_even, 2) # 2
    2
    """
    result = base
    while _____:
        digit = _____
        n = _____
        if _____ and _____:
            _____
    return result
```

(a) (6.0 pt)

```
def aggregate(n, func, cond, base):
    result = base
    while n > 0:
        digit = n % 10
        n = n // 10
        if cond(digit) and not digit == n % 10:
            result = func(result, digit)
    return result
```


6. (6.0 points) Forming Teams

Given a list of captain names `captains`, a list of player names `players`, and a two argument function `are_compatible`, the `form_teams` method returns a dictionary of compatible teams, where the key is the team captain and the value is a list of players on each team.

Compatibility between captains and players is determined by the `are_compatible` function, which returns `True` if a captain and a player are compatible and `False` otherwise. The order in which arguments are passed to this function does not matter, i.e. `are_compatible(p, c)` will return the same output as `are_compatible(c, p)`.

Players are added to the first captain they are compatible with. If players cannot be paired with any of the captains, they become a captain of their own solo team.

The current code has **3** different errors which you need to find!

```
def form_teams(captains, players, are_compatible):
    """
    >>> same_len = lambda x, y: len(x) == len(y)
    >>> captains = ['bob', 'mary', 'tim']
    >>> players = ['adam', 'kit', 'katie', 'margaret', 'tony', 'dory']
    >>> form_teams(captains, players, same_len)
    {'bob': ['kit'], 'mary': ['adam', 'tony', 'dory'], 'tim': [], 'katie': [], 'margaret': []}
    >>> same_first_or_last_char = lambda x, y: x[0] == y[0] or x[-1] == y[-1]
    >>> form_teams(captains, players, same_first_or_last_char)
    {'bob': [], 'mary': ['margaret', 'tony', 'dory'], 'tim': ['adam'], 'kit': [], 'katie': []}
    """
```

```
1. teams = {}
2. for c in captains:
3.     teams[c] = []
4. for p in players:
5.     added_to_team = False
6.     for c in captains:
7.         if are_compatible(p, c):
8.             teams[c] = [p]
9.             added_to_team = True
10.        else:
11.            added_to_team = False
12.        if not added_to_team:
13.            teams[p] = []
14. return teams
```

In each box, identify and fix each of the errors. Specify the line number(s) you would modify or delete for each error, and in the case of modification, give the modified line of code. You may not add new lines of code.

Use the following formats for your answer:

- To delete a line: delete line #
- To modify a line: line #: new line of code

(a) (2.0 pt)

Line 7 should be `if are_compatible(p, c)` and not `added_to_team`.
 Alternate fix: add `break` after line 9 within the `if are_compatible(p, c)` condition block

(b) (2.0 pt)

Line 9 should be `teams[c].append(p)` or `teams[c] += [p]`

(c) (2.0 pt)

Delete lines 10 and 11.

7. (5.0 points) Closet Overhaul

You've designed a closet abstract data type to help you organize your wardrobe.

A closet contains two things:

- **owner**: the name of the closet owner represented as a string
- **clothes**: the collection of clothes in the closet represented as a dictionary, where the key is the clothing item name and the value is the number of times the clothing item has been worn.

The `make_closet` constructor takes in `owner` (a string) and `clothes` (a list of strings representing clothing items) and returns a closet ADT.

Given this, you've implemented the abstract data type as follows:

```
def make_closet(owner, clothes):
    """ Create and returns a new closet. """
    clothes_dict = {}
    for item in clothes:
        clothes_dict[item] = 0
    return (owner, clothes_dict)

def get_owner(closet):
    """ Returns the owner of the closet """
    return closet[0]

def get_clothes(closet):
    """ Returns a dictionary of the clothes in the closet """
    return closet[1]
```

Given the closet ADT, implement the functions `wear_clothes` and `favorite_clothing_item`. You may not need all the lines provided, and you may need to change the indentation for some lines.

- (a) **(3.0 pt)** Implement `wear_clothes`, which takes a closet `closet` and a list of clothing items `clothes_worn`, and increments the number of times each item is worn by 1. If the clothing item specified does not already exist in the closet, add it to the closet.

```
def wear_clothes(closet, clothes_worn):
    """ Updates the number of times each clothing item is worn.
    >>> adam_closet = make_closet("adam", ["polo", "tie", "shorts"])
    >>> wear_clothes(adam_closet, ["shorts", "tie", "shorts"])
    >>> get_clothes(adam_closet)
    {'polo': 0, 'tie': 1, 'shorts': 2}
    >>> wear_clothes(adam_closet, ["polo", "scarf"])
    >>> get_clothes(adam_closet)
    {'polo': 1, 'tie': 1, 'shorts': 2, 'scarf': 1}
    """
```

```
def wear_clothes(closet, clothes_worn):
    clothes = get_clothes(closet)
    for c in clothes_worn:
        if c not in clothes:
            clothes[c] = 0
        clothes[c] += 1
```

- (b) (4.0 pt) Implement `favorite_clothing_item`, which takes in a closet `closet` and returns the name of the most frequently worn clothing item. Assume there are no ties.

```
def favorite_clothing_item(closet):  
    """ Finds the most frequently worn clothing item in a closet  
    >>> adam_closet = make_closet("adam", ["polo", "tie", "shorts"])  
    >>> wear_clothes(adam_closet, ["shorts", "tie", "shorts"])  
    >>> favorite_clothing_item(adam_closet)  
    'shorts'  
    >>> wear_clothes(adam_closet, ["tie", "polo", "polo", "scarf", "polo"])  
    >>> favorite_clothing_item(adam_closet)  
    'polo'  
    """  
    return max(_____, key = _____)
```

```
def favorite_clothing_item(closet):  
    return max(get_clothes(closet), key = lambda item: get_clothes(closet)[item])
```

8. (6.0 points) One At A Time

- (a) (6.0 pt) Implement the function `add_to_digit` which takes in two positive integers, `n` and `x`, and returns a new number that is the result of adding `x` to each digit of `n`. If the sum of `x` and a digit of `n` is more than 9, let the new digit be the rightmost digit of the sum.

```
def add_to_digit(n, x):
    """
    >>> add_to_digit(9, 5)      # 9 + 5 = 14
    4
    >>> add_to_digit(123, 5)   # 1 + 5 = 6, 2 + 5 = 7, 3 + 5 = 8
    678
    >>> add_to_digit(238, 5)   # 2 + 5 = 7, 3 + 5 = 8, 8 + 5 = 13
    783
    """
    new_digit = -----
    if -----:
        return -----
    return ----- * 10 + -----
```

```
def add_to_digit(n, x):
    new_digit = (n + x) % 10
    if n // 10 == 0:
        return new_digit
    return add_to_digit(n // 10, x) * 10 + new_digit
```

9. (8.0 points) Cart Tracker

You are the owner of several “smart” grocery stores. In order to model the supply of your shopping carts, complete the class `CartTracker` which allows you to keep track of a line of shopping carts at different stores!

A cart is represented as a `Cart` object, using the `Cart` class defined below.

```
class Cart:
    counter = 0
    def __init__(self, store):
        self.store = store # name of store where this cart is located
        self.id = Cart.counter
        Cart.counter += 1
```

```
class CartTracker:
    """
    >>> joes_store = CartTracker(3, 'Joes') # Carts 0 1 2
    >>> sally = joes_store.checkout_cart()
    >>> beth = joes_store.checkout_cart()
    >>> print(sally.id, beth.id)
    2 1
    >>> joes_store.add_cart(sally)
    >>> anna = joes_store.checkout_cart()
    >>> elsa = joes_store.checkout_cart()
    >>> print(anna.id, elsa.id)
    2 0
    >>> joes_store.checkout_cart()
    -1
    >>> ralphs_store = CartTracker(4, 'Ralphs') # Carts 3 4 5 6
    >>> CartTracker.transfer_carts(ralphs_store, joes_store, 2)
    >>> noah = joes_store.checkout_cart()
    >>> reese = ralphs_store.checkout_cart()
    >>> print(noah.id, reese.id)
    5 4
    >>> noah.store
    'Joes'
    """
    def __init__(self, num_carts, store):
        self.carts = -----
        self.store = -----

    def checkout_cart(self):
        if -----:
            return -1
        return -----

    def add_cart(self, cart):
        -----

    def transfer_carts(tracker1, tracker2, n):
        for i in range(-----):
            curr_cart = tracker1.-----
            curr_cart.store = -----
            -----
```

- (a) (2.0 pt) First complete the `__init__` method which takes in an integer `num_carts` and assigns the instance attribute `carts` to a list of `Cart` objects of length `num_carts`, representing a line of shopping carts.

The `id` attributes of the `Cart` objects in the list should be increasing from left to right. For example if `num_carts` is 3, then the list might look like [`<Cart 0>`, `<Cart 1>`, `<Cart 2>`] where the integers 0, 1, 2 are the `id` values for each cart respectively. Crucially the rightmost element, `<Cart 2>`, is the cart at the “front” of the line.

```
def __init__(self, num_carts, store):
    self.carts = [Cart(store) for i in range(num_carts)]
    self.store = store
```

- (b) (3.0 pt) Next, complete the following two methods so customers can checkout and put back shopping carts:

- `checkout_cart`: returns -1 if there are no carts available and otherwise removes the `Cart` object at the front of the line and returns it.
- `add_cart`: takes in a `Cart` object called `cart` and places it at the front of the shopping cart line (the right end of the list).

```
def checkout_cart(self):
    if len(self.carts) == 0:
        return -1
    return self.carts.pop()

def add_cart(self, cart):
    self.carts.append(cart)
```

- (c) (3.0 pt) Complete the class method `transfer_carts` which takes in two `CartTracker` objects representing different stores, `tracker1` and `tracker2`, and an integer `n`. `transfer_carts` removes `n` carts from `tracker1` by checking them out and adds them to `tracker2` in the same order they were checked out. Each transferred `Cart` object’s `store` attribute should be updated since the cart now belongs to a new store.

```
def transfer_carts(tracker1, tracker2, n):
    for i in range(n):
        curr_cart = tracker1.checkout_cart()
        curr_cart.store = tracker2.store
        tracker2.add_cart(curr_cart)
```

No more questions.