CS 61A Structure and Interpretation of Computer Programs
Spring 2021

MIDTERM 1

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.

Preliminaries

You can complete and submit these questions before the exam starts.

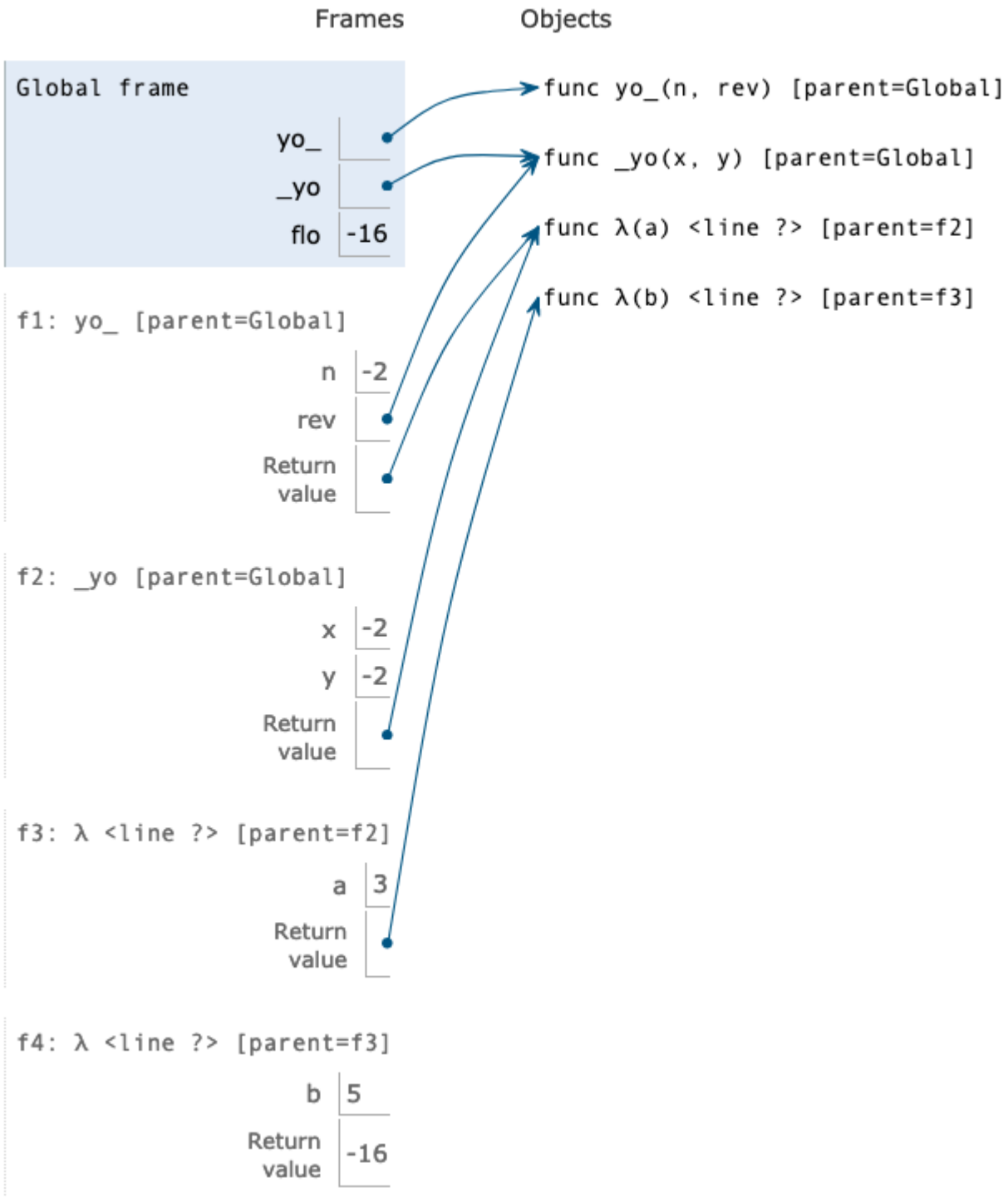
(a) What is your full name?

(b) What is your student ID number?

(c) By writing my name below, I pledge on my honor that I will abide by the rules of this exam and will neither give nor receive assistance. I understand that doing otherwise would be a disservice to my classmates, dishonor me, and could result in me failing the class.

1. (a) (8.0 points) Flow that Yo-Yo

The following environment diagram was generated by a program:



Click here to open the diagram in a new window

In this series of questions, you'll fill in the blanks of the program that follows so that its execution matches the environment diagram. You may want to fill in the blanks in a different order; feel free to answer the questions in whatever order works for you.

```
def yo_(n, rev):
    if n < 0:
        return ______
                 (a)
    elif n == 0:
        return float("inf")
    return n * -2

def _yo(x, y):
    if _______:
         (b)
        y += 1
    if _______:
         (c)
        return lambda a: ______
                              (d)
    return lambda a: lambda b: a + b

flo = yo_(_____, _____)(3)(5)
           (e)    (f)
```

(1.0 pt) Which of these could fill in blank (a)?

- ⓘ `lambda a: a - 3`
- ◯ `lambda a: rev(a, 3)(-1)`
- ◯ `rev`
- ◯ `yo_(rev)`
- ◯ `yo_`
- ◯ `rev(n, -3)`
- ◯ `lambda a: yo_(rev)`
- ◯ `_yo`

ii. (2.0 pt) Which of these could fill in blank (b)? *Select all that apply.*

- ☐ `y < 0`
- ☐ `x <= 0`
- ☐ `x == y`
- ☐ `y <= 0`
- ☐ `y == 0`
- ☐ `x < 0`

iii. (2.0 pt) Which of these could fill in blank (c)? *Select all that apply.*

- ☐ `x > 0`
- ☐ `x == -y`
- ☐ `y == -x`
- ☐ `x == y`
- ☐ `y > 0`
- ☐ `x < 0 and y < 0`

iv. (1.0 pt) Which of these could fill in blank (d)?

- ◯ `x + y`
- ◯ `lambda b: a**x + b**y`
- ◯ `a*x + b*y`
- ◯ `lambda b: a + b`
- ◯ `x * y`
- ◯ `lambda b: a*x * b*y`
- ◯ `a**x + b**y`
- ◯ `lambda b: a*x + b*y`

v. (1.0 pt) Which of these could fill in blank (e)?

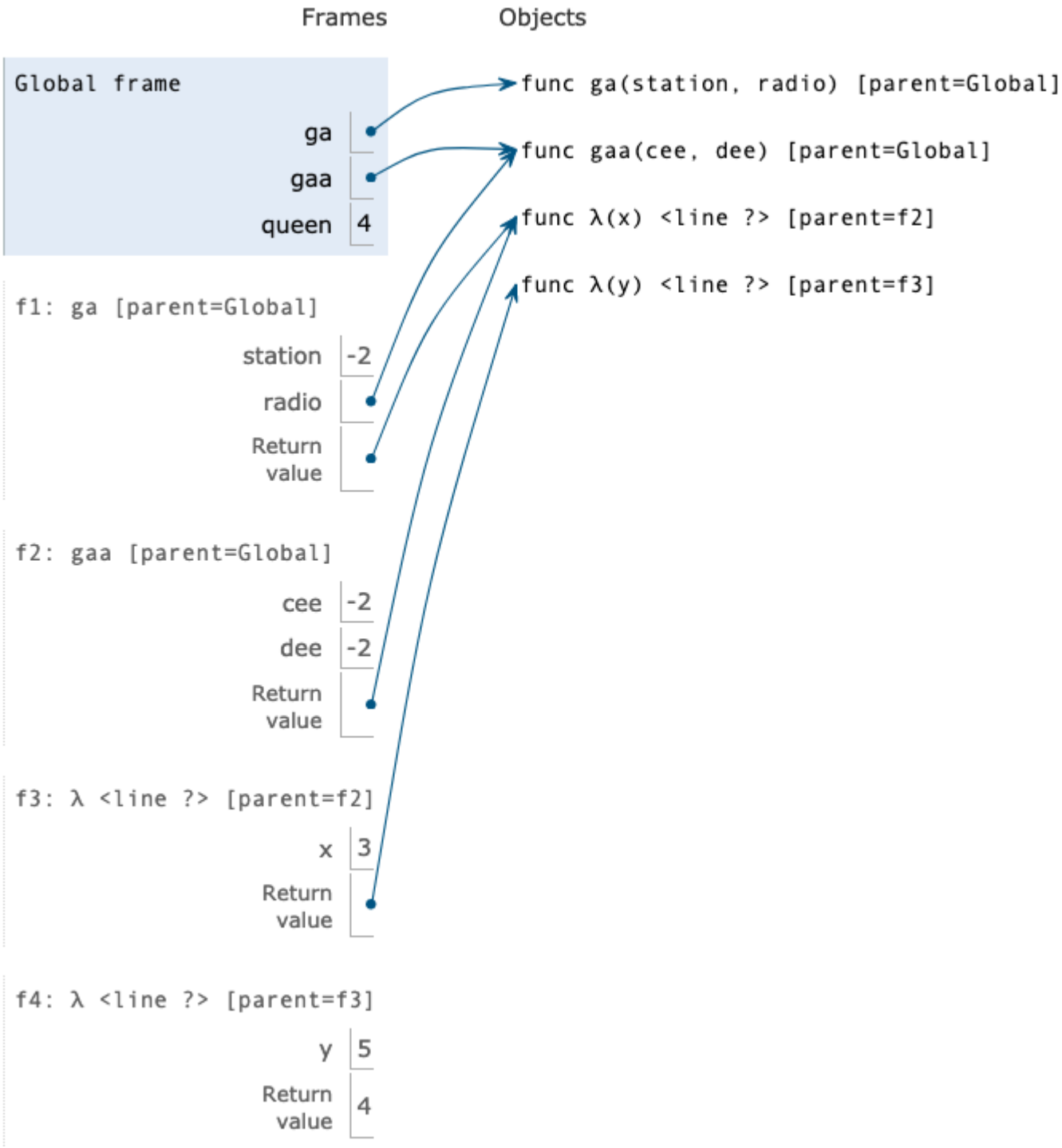
- ◯ `-4`
- ◯ `_yo`
- ◯ `rev`
- ◯ `-2`
- ◯ `yo_`
- ◯ `-3`

vi. (1.0 pt) Which of these could fill in blank (f)?

- ◯ `yo_`
- ◯ `rev`
- ◯ `lambda x: x`
- ◯ `_yo`
- ◯ `lambda x: _yo(x)`
- ◯ `flo`

(8.0 points) Radio Ga-Ga

The following environment diagram was generated by a program:



Click here to open the diagram in a new window

In this series of questions, you'll fill in the blanks of the program that follows so that its execution matches the environment diagram. You may want to fill in the blanks in a different order; feel free to answer the questions in whatever order works for you.

```
def ga(station, radio):
    if station < 0:
        return ______
                 (a)
    elif station == 0:
        return float("inf")
    return station * -98

def gaa(cee, dee):
    if _______:
         (b)
        dee += 1
    if _______:
         (c)
        return lambda x: ______
                              (d)
    return lambda x: lambda y: x * y

queen = ga(_____, _____)(3)(5)
           (e)    (f)
```

(1.0 pt) Which of these could fill in blank (a)?

(b) i.

- ◯ `lambda x: station + x - 3`
- ◯ `lambda x: radio(station, 3)(-1)`
- ◯ `radio`
- ◯ `ga(radio)`
- ◯ `ga`
- ◯ `radio(station, -3)`
- ◯ `lambda x: ga(gaa)`
- ◯ `gaa`

ii. (2.0 pt) Which of these could fill in blank (b)? *Select all that apply.*

- ☐ `cee < 0`
- ☐ `dee <= 0`
- ☐ `cee == dee`
- ☐ `cee <= 0`
- ☐ `cee == 0`
- ☐ `dee < 0`

iii. (2.0 pt) Which of these could fill in blank (c)? *Select all that apply.*

- ☐ `cee > 0`
- ☐ `cee == -dee`
- ☐ `dee == -cee`
- ☐ `cee == dee`
- ☐ `dee > 0`
- ☐ `cee < 0 and dee < 0`

iv. (1.0 pt) Which of these could fill in blank (d)?

- ◯ `cee - dee`
- ◯ `lambda y: x * cee - y * dee`
- ◯ `cee * x - dee * x`
- ◯ `lambda y: cee + y`
- ◯ `cee * dee`
- ◯ `lambda y: cee * x * dee * y`
- ◯ `cee**x + dee**y`
- ◯ `lambda y: x * cee + y * dee`

v. (1.0 pt) Which of these could fill in blank (e)?

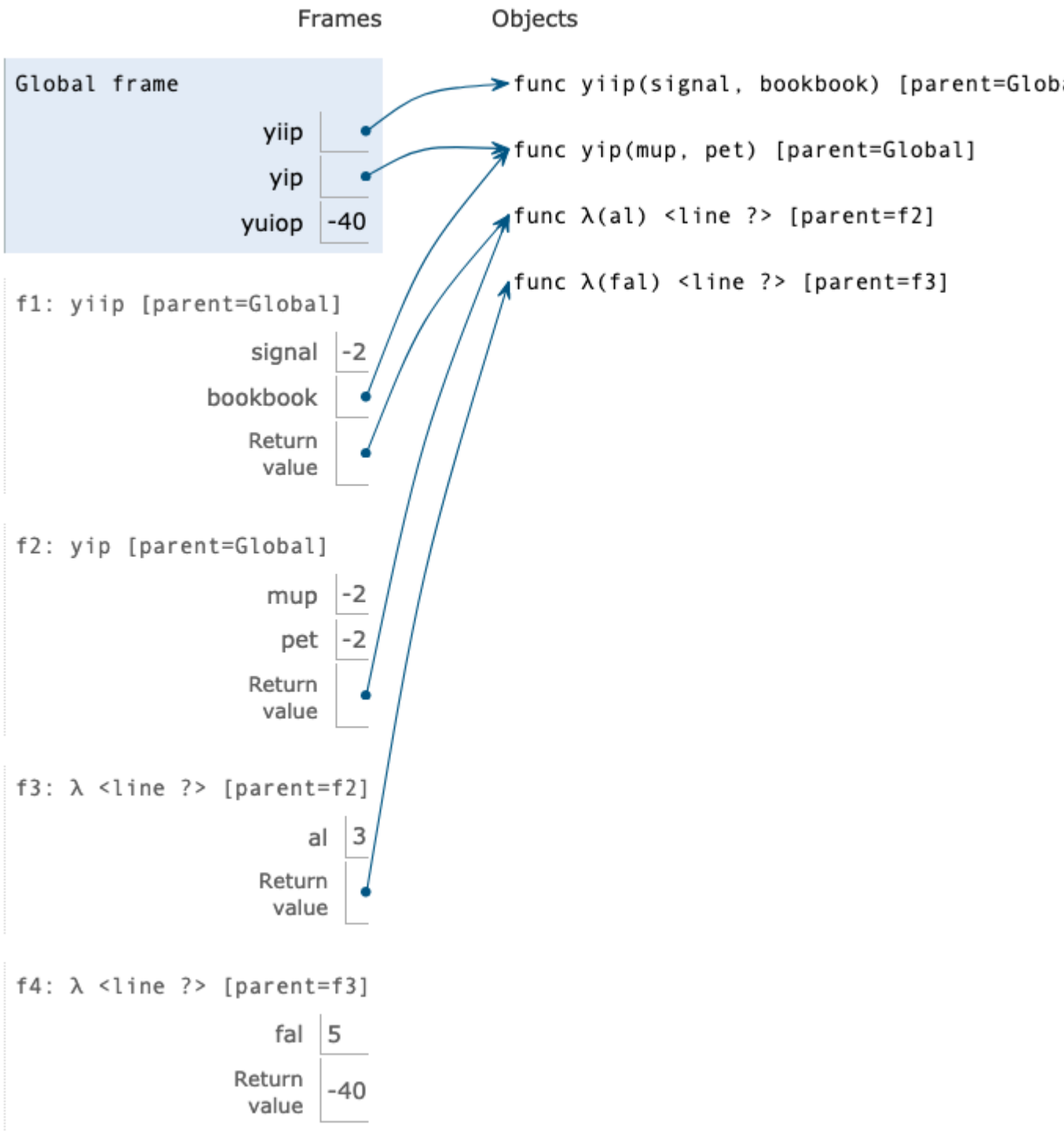
- ◯ `gaa(-2)`
- ◯ `gaa`
- ◯ `radio(ga)`
- ◯ `-2`
- ◯ `ga`
- ◯ `station`

vi. (1.0 pt) Which of these could fill in blank (f)?

- ◯ `ga`
- ◯ `ga()`
- ◯ `lambda n: n`
- ◯ `gaa`
- ◯ `gaa()`
- ◯ `lambda n: gaa(n)`

(8.0 points) YipYip Book

The following environment diagram was generated by a program:



Click here to open the diagram in a new window

In this series of questions, you'll fill in the blanks of the program that follows so that its execution matches the environment diagram. You may want to fill in the blanks in a different order; feel free to answer the questions in whatever order works for you.

```
def yiip(signal, bookbook):
    if signal < 0:
        return ______
                 (a)
    elif signal == 0:
        return float("inf")
    return signal * -98

def yip(mup, pet):
    if _______:
         (b)
        mup += 1
    if _______:
         (c)
        return lambda al: ______
                              (d)
    return lambda al: lambda fal: al - fal

yuiop = yiip(_____, _____)(3)(5)
               (e)    (f)
```

(1.0 pt) Which of these could fill in blank (a)?

(c) i.
- ◯ `lambda al: signal - al * 3`
- ◯ `lambda al: bookbook(signal, 3)(al)`
- ◯ `bookbook`
- ◯ `yiip(bookbook)`
- ◯ `yip`
- ◯ `bookbook(-3, signal)`
- ◯ `bookbook(signal + - 3)`

ii. (2.0 pt) Which of these could fill in blank (b)? *Select all that apply.*

- ☐ `mup < 0`
- ☐ `pet <= 0`
- ☐ `mup == pet`
- ☐ `mup <= 0`
- ☐ `pet == 0`
- ☐ `pet < 0`

iii. (2.0 pt) Which of these could fill in blank (c)? *Select all that apply.*

- ☐ `mup > 0`
- ☐ `mup == -pet`
- ☐ `pet == -mup`
- ☐ `mup == pet`
- ☐ `pet > 0`
- ☐ `mup <= 0 and pet <= 0`

iv. (1.0 pt) Which of these could fill in blank (d)?

- ◯ `al - fal**fal`
- ◯ `lambda fal: mup**al + pet**fal`
- ◯ `mup * al + pet * fal`
- ◯ `lambda fal: mup + pet * fal`
- ◯ `mup * pet`
- ◯ `lambda fal: mup * al * pet * fal`
- ◯ `mup**al + pet**al`
- ◯ `lambda fal: al * mup + fal * pet`

v. (1.0 pt) Which of these could fill in blank (e)?

- ◯ `yiip(2 * -1)`
- ◯ `yiip`
- ◯ `bookbook(yip)`
- ◯ `-2`
- ◯ `yip`
- ◯ `signal - 2`

vi. (1.0 pt) Which of these could fill in blank (f)?

- ◯ `yip`
- ◯ `yip()`
- ◯ `lambda y: y`
- ◯ `yiip`
- ◯ `yiip()`
- ◯ `lambda y: yiip(y)`
- ◯ `-2`

2. (1.0 points) The Case of the Missing Docstring

Consider the following function and its doctests:

```
def mystery(l):
  """
  >>> mystery([1, 2, 3, 4])
  2.5
  >>> mystery([2, 4])
  3.0
  >>> mystery([-5, -2, -9])
  0
  >>> mystery([])
  0
  >>> mystery([345])
  345.0
  """
  s = 0
  t = 0
  for item in l:
    if item > 0:
      s += item
      t += 1
  if t == 0:
    return 0
  return s/t
```

(a) Which of the following docstrings would best describe that function?

◯ Returns the average of all elements in L or returns zero if no elements exist.

◯ Returns the average of elements in L with an odd index or returns zero if no such elements exist.

◯ Returns the average of elements in L that are >= 0 or returns zero if no such elements exist.

◯ Returns the average of positive elements in L or returns zero if no such elements exist.

3. (2.0 points) Magical Test Weaver

Consider the following function signature and docstring:

```
def magic_weave(a, b, c):
    """
    Assuming A and B are positive integers with the same number of base-10 digits
    and C is a positive integer < 10, return the number whose base-10
    representation is the interleaving of digits in A and B (alternating
    first one from A then one from B) from all positions where the
    two digits in A and B at that position are both >= C.  Return 0 if there
    are no such positions. Raises an exception if preconditions are not met.
    """
```

Here is one example of a passing doctest:

```
>>> magic_weave(345, 987, 3)
394857
```

(a) Based on the docstring of that function, which of these would be passing doctests? *Select all that apply.*

☐
```
>>> magic_weave(123, 456, 5)
56
```

☐
```
>>> magic_weave(234, 456, 5)
3546
```

☐
```
>>> magic_weave(456, 567, 5)
5667
```

☐
```
>>> magic_weave(0, 0, 5)
0
```

☐
```
>>> magic_weave(101, 202, 0)
120012
```

☐
```
>>> magic_weave(567, 899, 10)
586979
```

4. (10.0 points) Domain On the Range

The domain of a function is the set of all possible argument values, while the range of a function is the set of values that it can return. In this two part-question, you will implement higher-order functions to restrict the domain and range of other functions.

(a) (4.0 points) restrict_domain

Implement `restrict_domain`, a function that accepts three parameters (`f`, `low_d`, `high_d`) and returns a higher-order function that returns the same thing as `f` when given an argument between `low_d` and `high_d`, inclusive, and otherwise returns `float("-inf")`.

```
def restrict_domain(f, low_d, high_d):
    """Returns a function that restricts the domain of F,
    a function that takes a single argument x.
    If x is not between LOW_D and HIGH_D (inclusive),
    it returns -Infinity, but otherwise returns F(x).

    >>> from math import sqrt
    >>> f = restrict_domain(sqrt, 1, 100)
    >>> f(25)
    5.0
    >>> f(-25)
    -inf
    >>> f(125)
    -inf
    >>> f(1)
    1.0
    >>> f(100)
    10.0
    """
    ---------
    #   (a)
        ---------
        #   (b)
            ---------
            #  (c)
        ---------
        #  (d)
    return wrapper_method_name
```

i. (1.0 pt) Fill in blank (a).

ii. (1.0 pt) Fill in blank (b).

iii. (1.0 pt) Fill in blank (c).

iv. (1.0 pt) Fill in blank (d).

(b) (5.0 points) restrict_range

Implement `restrict_range`, a function that accepts three parameters (`f`, `low_r`, `high_r`) and returns a higher-order function that returns the same thing as `f` when that result is between `low_r` and `high_r` (inclusive), and otherwise returns `float("-inf")`.

```
def restrict_range(f, low_r, high_r):
   """Returns a function that restricts the range of F, a function
   that takes a single argument X. If the return value of F(X)
   is not between LOW_R and HIGH_R (inclusive), it returns -Infinity,
   but otherwise returns F(X).

   >>> cube = lambda x: x * x * x
   >>> f = restrict_range(cube, 1, 1000)
   >>> f(1)
   1
   >>> f(-5)
   -inf
   >>> f(5)
   125
   >>> f(10)
   1000
   >>> f(11)
   -inf
   """
    ---------
    #   (a)
        ---------
        #   (b)
        ---------
        #   (c)
            ---------
            #  (d)
        ---------
        #  (e)
    return wrapper_method_name
```

i. (1.0 pt) Fill in blank (a).

ii. (1.0 pt) Fill in blank (b).

iii. (1.0 pt) Fill in blank (c).

iv. (1.0 pt) Fill in blank (d).

v. (1.0 pt) Fill in blank (e).

(c) (1.0 points) restrict_both

Now that you have those two functions defined, you'll implement `restrict_both`, a higher-order function that accepts a function `f` and four numeric arguments (`low_d`, `high_d`, `low_r`, `high_r`) and returns the result of applying both `restrict_domain` and `restrict_range` on `f`.

```
def restrict_both(f, low_d, high_d, low_r, high_r):
    """
    Returns a version of F with a domain restricted to (LOW_D, HIGH_D)
    and a range restricted to (LOW_R, HIGH_R).

    >>> diva = lambda x: (10000 // x) * 9
    >>> f = enforce_both(diva, 1, 1000, 100, 999)
    >>> f(0)
    -inf
    >>> f(10000)
    -inf
    >>> f(200)
    450
    >>> f(100)
    900
    >>> f(1000)
    -inf
    """
    -------------
```

i. Fill in the blank. You may use one line or multiple lines, as long as the solution is correct. Your solution should use `restrict_domain` and `restrict_range` somehow, and assume that those were implemented correctly.

If you wrote your code in code.cs61a.org, you can paste it in here (do not worry if it seems too indented, as long as the indentation worked there).

5. (8.0 points) Digit replacer

The function `digit_replacer(predicate, transformer)` should return a function that replaces all the digits in a number where `predicate(digit)` is true with the result of `transformer(digit)`. The returned function should accept a single argument, the number `n`, and return the number with the digits replaced.

Here is the function signature and doctests:

```
def digit_replacer(predicate, transformer):
    """Returns a function that accepts a single number N (where N > 0) and
    returns a number where all digits that return true for PREDICATE(DIGIT)
    have been replaced by TRANSFORMER(DIGIT). TRANSFORMER is assumed to always
    return a valid digit >= 0 and <= 9.

    >>> is_even = lambda d: d % 2 == 0
    >>> lt_five = lambda d: d < 5
    >>> always_two = lambda d: 2
    >>> floor_divide_two = lambda d: d // 2
    >>> digit_replacer(is_even, floor_divide_two)(21098)
    11094
    >>> digit_replacer(lt_five, always_two)(1064592)
    2262592
    """
```

(a) (4.0 pt) Use iteration (**without recursion**) to implement `digit_replacer`.

Your solution should only use numbers, arithmetic expressions, and booleans. It should *not* use strings, lists, or other data types, and will earn 0 points if it does. It will also earn 0 points if it uses recursion, since that's tested in the second part of this question.

Here's an approximate structure of a solution, if that helps guide your implementation. It does not necessarily reflect the exact number of lines or indentation.

```
def digit_replacer(predicate, transformer):
    _____
        _____
        _____

        while ______:
            _____
                _____
            _____
                _____
            _____
            _____
       return _____
    ______
```

Remember that you can use code.cs61a.org to try out your code and see if it passes the doctests. You can then paste the code here. Please include the function signature.

(b) (4.0 pt) Use a recursive approach to implement `digit_replacer`.

Your solution should only use numbers, arithmetic expressions, and booleans. It should *not* use strings, lists, or other data types, and will earn 0 points if it does. It will also earn 0 points if it does not use recursion, since that was tested in the first part of this question.

Here's an approximate structure of a solution, if that helps guide your implementation. It does not reflect the exact number of lines or indentation.

```
def digit_replacer(predicate, transformer):
    ------
        ------
            ------
        ------
            ------
        ------
       return ------
    ------
```

Remember that you can use code.cs61a.org to try out your code and see if it passes the doctests. You can then paste the code here. Please include the function signature.

6. (6.0 points) Run checker

Let's use the term "chain function" to mean a function that takes a single numerical argument and returns another chain function (so that if `h` is a chain function, one can call `h(3)(4)(1)(2)(0)...`, which we'll call a "chain of calls.")

The function `run_checker` accepts two functions as arguments (`condition` and `result`) and returns a chain function. Each call in a chain starting with the function returned by `run_checker` first applies `condition` to the two previous arguments in the chain and its own argument. If that returns a false value, it prints "No run!" and otherwise prints the result of `result` applied to the same three arguments. For the first and second calls in the chain, the missing arguments are taken to be -1.

```
def run_checker(condition, result):
    """
    Returns a chain function. Each call in a chain that starts with
    this returned function prints "No run!" if CONDITION returns a false
    value when applied to the previous two arguments and the current argument,
    and otherwise prints the result of applying RESULT to these same
    three arguments.  For calls in the chain where there are fewer than two
    preceding calls in the chain, the missing arguments are taken to be -1.

    >>> f = run_checker(lambda a, b, c: a > b > c and a >= 10, lambda a, b, c: a*(b+c))
    >>> f = f(15)
    No run!
    >>> f = f(10)
    No run!
    >>> f = f(5)
    225
    >>> f = f(2)
    70
    >>> f = f(1)
    No run!
    >>> f = f(11)
    No run!
    >>> f = f(12)
    No run!
    >>> f = f(10)
    No run!
    >>> f = f(2)
    144
    """
    def f(______):
       #   (a)
        def g(______):
           #   (b)

            ------
            ------
            ------
            ------
            ------
            # (c)

        return g
    return f(______)
    #          (d)
```

(a) **(1.0 pt)** Fill in blank (a).

(b) **(1.0 pt)** Fill in blank (b).

(c) **(3.0 pt)** Fill in the (c) blanks. You can use more or fewer lines for your solution than the number suggested by the blanks.

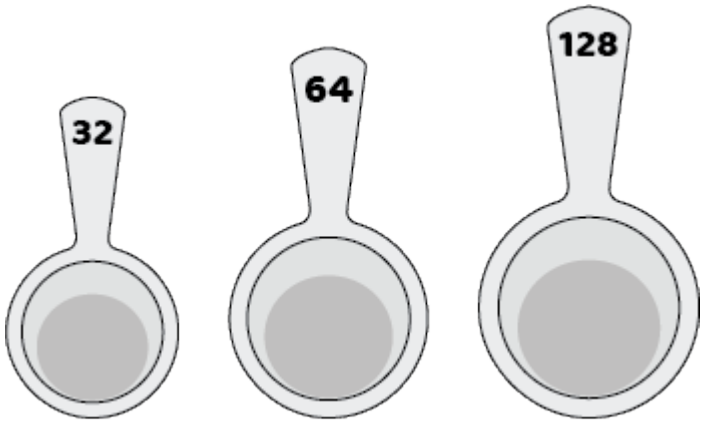
If you use code.cs61a.org to try out your code, you can then paste the code here. Don't worry if it seems too indented, as long as the indentation worked there.

(d) **(1.0 pt)** Fill in blank (d).

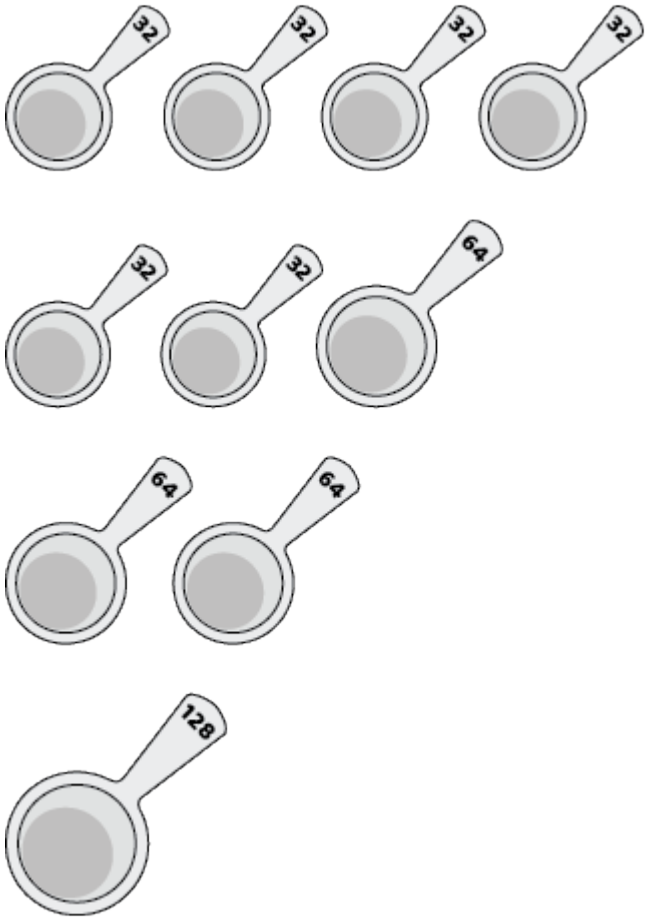
7. (5.0 points) Measure Twice, Cup Once

The function `measure_methods` accepts two arguments (`total_needed`, the total number of grams needed for a recipe, and `cup_sizes`, a list of measuring cup sizes available) and returns the number of possible ways to make exactly `total_needed` using the cup sizes. The `cup_sizes` list is measured in grams, sorted from smallest to largest, and each size is a power of 2. A cup size may be used multiple times in order to come up with `total_needed`.

For example, the list of `[32, 64, 128]` represents these cups:



There are 4 possible ways to make a total of 128 grams from those cups:



Here's the partially defined function:

```
def measure_methods(total_needed, cup_sizes):
```

```
    """Returns the number of ways to make exactly TOTAL_NEEDED with
    the given list of CUP_SIZES (sorted by smallest to largest).

    >>> measure_methods(128, [32, 64, 128])
    4
    >>> measure_methods(256, [32, 64, 128])
    9
    >>> measure_methods(384, [32, 64, 128])
    16
    >>> measure_methods(256, [16, 32, 64])
    25
    >>> measure_methods(125, [32, 64, 128])
    0
    """
    def helper_method_name(______):
        #        (a)
        ------
        ------
        ------
        ------
        ------
        # (b)
    return helper_method_name(______)
    #               (c)
```

(a) (1.0 pt) Fill in blank (a).

(b) (3.0 pt) Fill in the (b) blanks. You can use more or fewer lines for your solution than the number suggested by the blanks, and your code may involve multiple indentation levels.

If you use code.cs61a.org to try out your code, you can then paste the code here. Don't worry if it seems too indented, as long as the indentation worked there.

(c) (1.0 pt) Fill in blank (c).

8. (1.0 points) Extra Point!

(a) (1.0 pt) "Baked Century" (nine letters, two words).

No more questions.