# CS 61A Fall 2023

# Structure and Interpretation of Computer Programs

Midterm 1 Solutions

**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)** What is your @berkeley.edu email address?

**(d)** Sign (or type) your name to confirm that all work on this exam will be your own. The penalty for academic misconduct on an exam is an F in the course.

**1. (6.0 points) What Would Python Display?**

Assume the following code has been executed already.

```
cat = 3
def dog():
    return cat
def hog():
    cat = 4
    print(cat, dog())
cat = 5
```

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", but include all output displayed before the error. To display a function value, write "Function".

**(a) (1.0 pt)** `dog(cat)`

- ○ 3
- ○ 4
- ○ 5
- ○ Function
- ● Error

The function `dog` is defined to take no arguments, but this call expression passes an argument.

**(b) (1.0 pt)** `dog()`

- ○ 3
- ○ 4
- ● 5
- ○ Function
- ○ Error

The name cat is in the global frame. It is assigned to 5, which overwrites the assignment to 3.

Some might answer 3, as cat is initially bound to 3. But since `dog()` is being called after `cat` is bound to 5 and the body of `dog` is only evaluated when the function is called, it will return 5.

**(c) (2.0 pt)** `print((lambda dog: print(dog()))(lambda: 6))`

```
6
None
```

The entire expression is a call expression. The first lambda expression defines a function with an argument named `dog` and prints the result of calling dog with no arguments. That function is then called on a function defined by the second lambda which takes no arguments and always returns 6.

Thus, the result of calling the function `lambda: 6` is printed.

**(d) (2.0 pt)** `hog()`

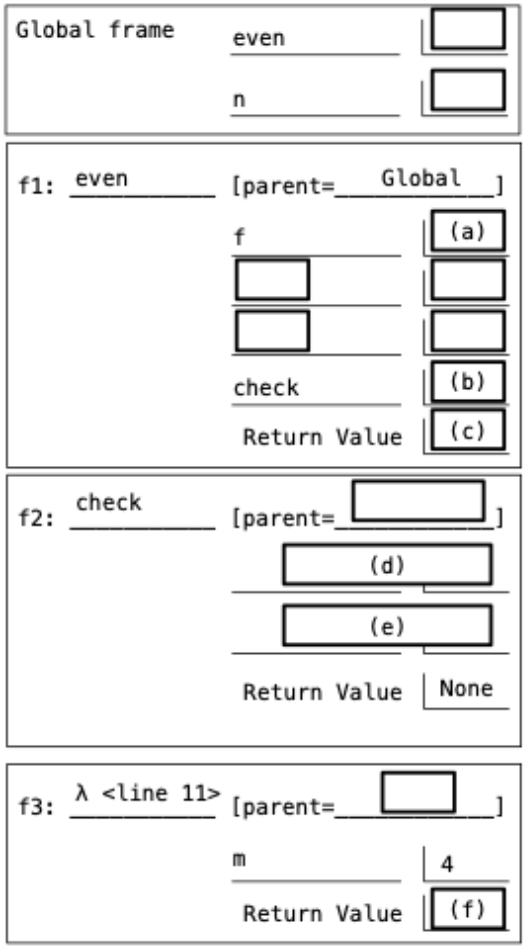
```
4 5
```

Two values are passed to `print`, the value of `cat` which appears in the local frame and is 4, as well as the result of calling `dog()` which is 5.

A key thing to note is that the environment of the call to dog() does not include the frame from the call to hog().

**2. (8.0 points) An Odd Implementation of Even**

Complete the environment diagram below and then answer the questions that follow. There is one question for each labeled blank in the diagram. The blanks with no labels have no questions associated with them and are not scored. **Some blanks may be empty or unused. If a blank contains an arrow to a function, write the function as it would appear in the diagram. Do not add frames for calls to built-in functions.**




```
 1: def even(f, n):
 2:     "Return whether f(n) is even."
 3:     even = False
 4:     def check(h):
 5:         if h(n) % 2 == 0:
 6:             even = True
 7:     check(f)
 8:     return even
 9:
10: n = 2
11: even(lambda m: m + n, 4)
```

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

- ● `func lambda(m) <line 11> [parent=Global]`
- ○ `func lambda(m) <line 11> [parent=f1]`
- ○ `func lambda(m, n) <line 11> [parent=Global]`
- ○ `func lambda(m, n) <line 11> [parent=f1]`

`lambda m: m + n` is not defined within another function definition. It is evaluated in the global frame.

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

- ○ `func check(f) [parent=Global]`
- ○ `func check(f) [parent=f1]`
- ○ `func check(h) [parent=Global]`
- ● `func check(h) [parent=f1]`

The function `check` is defined in the call to `even`, and thus its parent frame is the `even` frame.

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

- ○ `None`
- ○ A function named `check`
- ○ `True`
- ● `False`
- ○ Blank: an error occurs before a value is returned.

The name `even` is still bound to `False` because the assignment statement in line 6 makes a change in the `check` frame (f2), not the `even` frame (f1).

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

- ○ `f` is bound to a lambda function.
- ● `h` is bound to a lambda function.
- ○ `f` is bound to a number.
- ○ `h` is bound to a number.

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

- ○ `n` is bound to a number.
- ○ `m` is bound to a number.
- ○ `check` is bound to `True`
- ○ `check` is bound to `False`
- ● `even` is bound to `True`
- ○ `even` is bound to `False`
- ○ Blank: there is no name-value binding here.

Assignment always changes the current frame, and so line 6 adds a binding from `even` to `True` here.

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

- ○ 2
- ○ 4
- ● 6
- ○ 8
- ○ Blank: an error occurs before a value is returned.

The lambda frame's parent is the lambda function's parent, which is the global frame. Therefore, `n` evaluates to 2.

**(g) (2.0 pt)** What problems are there in the `even` function's implementation? **Select all that apply.**

- ☐ It does not call `f` on `n` for its original arguments `f` and `n`.
- ☐ It does not correctly test whether `f(n)` is an even number.
- ☐ It always stops without returning due to an error.
- ■ It always returns `False` regardless of the value of `f(n)`.
- ☐ None of the above, but there is some other error in the implementation.
- ☐ None of the above because there is no error in the implementation.

Since `even` in the `even` frame is never reassigned, the `even` function will always return `False`.

**3. (11.0 points) In Your Prime**

**Definition**: The *prime gap* of a prime number `p` is the difference `q-p` between `p` and the next larger prime `q`.

Assume a function `is_prime` is implemented that takes a positive integer and returns `True` if it is prime and `False` otherwise. You may call `is_prime` in your implementations below.

```
def is_prime(n):
    """Return whether positive integer n is a prime number."""
```

**(a) (6.0 points)**

Implement `smallest_gap`, which takes a positive integer `k`. It returns the smallest prime number that has a prime gap greater than or equal to `k`.

```
def smallest_gap(k):
    """Return the smallest prime number with prime gap greater than or equal to k.

    >>> smallest_gap(2) # 5 - 3 = 2
    3
    >>> smallest_gap(6) # 29 - 23 = 6
    23
    >>> smallest_gap(9) # 127 - 113 = 14, and no prime below 113 has a gap above 8
    113
    """
    p, q = 2, 3  # The two smallest prime numbers
    while _______:
            (a)
        p, q = _______
                 (b)
        while _______:
                (c)
            q = q + 1
    return p
```

**i. (2.0 pt)** Fill in blank (a).

- ○ `q - p == k`
- ○ `q - p != k`
- ○ `q - p >= k`
- ○ `q - p <= k`
- ○ `q - p > k`
- ● `q - p < k`

The while condition expresses when to keep looking.

**ii. (2.0 pt)** Fill in blank (b).

```
q, q + 1
```

This sets p to be the next prime q, and then initializes q to be the next number so that we can search for the next prime in the following while loop.

**iii. (2.0 pt)** Fill in blank (c). **You may not use `and` or `or`.**

```
not is_prime(q)
```

Keep incrementing q until it is prime.

**(b) (5.0 points)**

Implement `plus_one`, a function that takes two different positive integers `a` and `b`. If the larger one is prime, it returns one more than the smaller one. Otherwise, it returns one more than the average of `a` and `b`.

```
def plus_one(a, b):
    """Return one more than either the smaller (if larger is prime) or average of a and b.

    >>> plus_one(5, 7) # 7 is prime, so return 1 more than 5
    6
    >>> plus_one(15, 7) # 15 is not prime, so return 1 more than the average of 15 and 7
    12.0
    """
    assert a > 0 and b > 0 and a != b

    if _______:
         (d)

        f = _______
              (e)

    else:

        f = _______
              (f)

    return f(a, b) + 1
```

**i. (2.0 pt)** Fill in blank (d).

```
is_prime(max(a, b))
```

**ii. (1.0 pt)** Fill in blank (e).

- ○ `max`
- ● `min`
- ○ `max()`
- ○ `min()`
- ○ `max(a, b)`
- ○ `min(a, b)`

Since the return statement calls the function f, one should assign a function to f. It should be min in this case according to the spec as the return statement adds 1 to the min of a,b if f is min.

**iii. (2.0 pt)** Fill in blank (f).

```
lambda a, b: (a+b)/2
```

Since there is no built-in averaging function, one must be defined.

**4. (15.0 points) Choose Wisely**

**Definition**: A *digit test* is a function that takes a non-negative integer less than 10 and returns `True` or `False`.

**(a) (9.0 points)**

Implement `only` which takes a non-negative integer `n` and a *digit test* `t`. It returns a non-negative integer containing only the digits of `n` for which `t` returns `True` when called on each of those digits. The digits should appear in the same order as they did in `n`. The number 0 has no digits. **You may not use `str` or `repr` or `[` or `]` or `for`.**

```
def only(n, t):
    """Return only the digits of n for which t returns True when called on each digit.

    >>> only(23344567, lambda d: d % 2 == 0)
    2446
    >>> only(987654349675, lambda d: d < 7)
    6543465
    >>> only(2023, lambda d: False)
    0
    """
    keep = 0
    while n:
        n, d = n // 10, n % 10

        if _______:
             (a)

            keep = 10 * keep + _______
                                  (b)
    _______:
      (c)

        _______
          (d)
    return n
```

**i. (2.0 pt)** Fill in blank (a).

```
t(d)
```

Calls the digit test on the current digit to see if one should include that digit.

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

- (●) `d`
- ( ) `d // 10`
- ( ) `n`
- ( ) `n % 10`
- ( ) `keep`
- ( ) `keep % 10`
- ( ) `keep // 10`

**iii. (2.0 pt)** Fill in blank (c)

- ○ `if True`
- ○ `if n`
- ○ `if keep`
- ○ `if n > keep`
- ○ `if keep > n`
- ○ `while True`
- ○ `while n`
- ● `while keep`
- ○ `while n > keep`
- ○ `while keep > n`

**iv. (4.0 pt)** Fill in blank (d) **with an assignment statement**. You are allowed to assign to multiple names.

```
n, keep = n * 10 + keep % 10, keep // 10
```

**(b) (6.0 points)**

Implement `every` which takes a *digit test* `t` and returns a function `digit` that takes a positive integer `n`. The `digit` function returns whether `t` returns True for every digit of `n`.

```
def every(t):
    """Return a function of n that returns whether t returns True for every digit of n.

    >>> f = every(lambda d: d % 2 == 1)
    >>> f(37511)  # every digit is odd
    True
    >>> f(2023)   # Not every digit is odd
    False
    """
    def digit(n):
        assert n > 0
        while n:
            if _______:
                 (e)
                _______
                  (f)
            n = n // 10
        return _______
                  (g)
    return _______
             (h)
```

**i. (3.0 pt)** Fill in blank (e).

```
not t(n % 10)
```

**ii. (1.0 pt)** Fill in blank (f).

```
return False
```

**iii. (1.0 pt)** Fill in blank (g).

- ○ `every`
- ○ `digit`
- ● `True`
- ○ `False`
- ○ `n > 0`

**iv. (1.0 pt)** Fill in blank (h).

- ● `digit`
- ○ `digit()`
- ○ `digit(n)`
- ○ `lambda n: digit`

**(c) This A+ question is not worth any points. It can only affect your course grade if you have a high A and might receive an A+. Finish the rest of the exam first!**

Implement `even_odd`, which takes a positive integer `n` that has both even and odd digits. It returns `True` if all of the odd digits of `n` are larger than the last (right-most) even digit of `n`. Otherwise, it returns `False`. You may call `only` and `every`. **You may not use `str` or `repr` or `[` or `]` or `for`.**

```
def even_odd(n):
    """Return whether all odd digits of n are larger than n's last even digit.

    >>> even_odd(8023) # 3 (the only odd digit) is larger than 2 (the last even digit)
    True
    >>> even_odd(5237679) # 3 (an odd digit) is smaller than 6 (the last even digit)
    False
    >>> even_odd(7297679) # All odd digits (7 & 9) are larger than 6 (the last even digit)
    True
    """
    return _______
```

**i.** Fill in the blank. You may only write one return expression, but if it's long you can use multiple lines.

```
every(lambda d: d % 2 == 0 or d > only(n, lambda d: d % 2 == 0) % 10)(n)
every(lambda d: d > only(n, lambda d: d % 2 == 0) % 10)(only(n, lambda
d: d % 2 == 1))
```

**No more questions.**