CS 61A Structure and Interpretation of Computer Programs

Spring 2023 MIDTERM 1

INSTRUCTIONS

You have 1 hour and 50 minutes to complete the exam.

- The exam is closed book, closed notes, closed computer, closed calculator, except one 8.5" × 11" page of your own creation and the provided midterm 1 study guide.
- Mark your answers on the exam itself in the spaces provided. We will not grade answers written on scratch paper or outside the designated answer spaces.
- If you need to use the restroom, bring your phone and exam to the front of the room.
- You may use built-in Python functions that do not require import, such as `pow`, `len`, `abs`, `bool`, `int`, `float`, `str`, `round`, `max`, and `min`.
- You **may not** use example functions defined on your study guide unless a problem clearly states you can.
- Unless otherwise specified, you are allowed to reference functions defined in previous parts of the same question.
- You may not use ; to place two statements on the same line.

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. (10.0 points) What Would Python Display?

Assume the following code has been executed.

```
e = 40
def glob(e):
    print(e)
    e = e + 1
    return e

def mid(nights):
    days = 7
    if days or nights:
        days, nights = (days and days + 1), (nights or days)
    print(days)
    return nights

def special(f, x):
    return f(x) + f(x)
```

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) (2.0 pt) `print(0, print(1), 2)`

1
0 None 2

(b) (4.0 pt) `print(glob(e), glob(3), e)`

40
3
41 4 40

(c) (2.0 pt) `mid(not 'swift')`

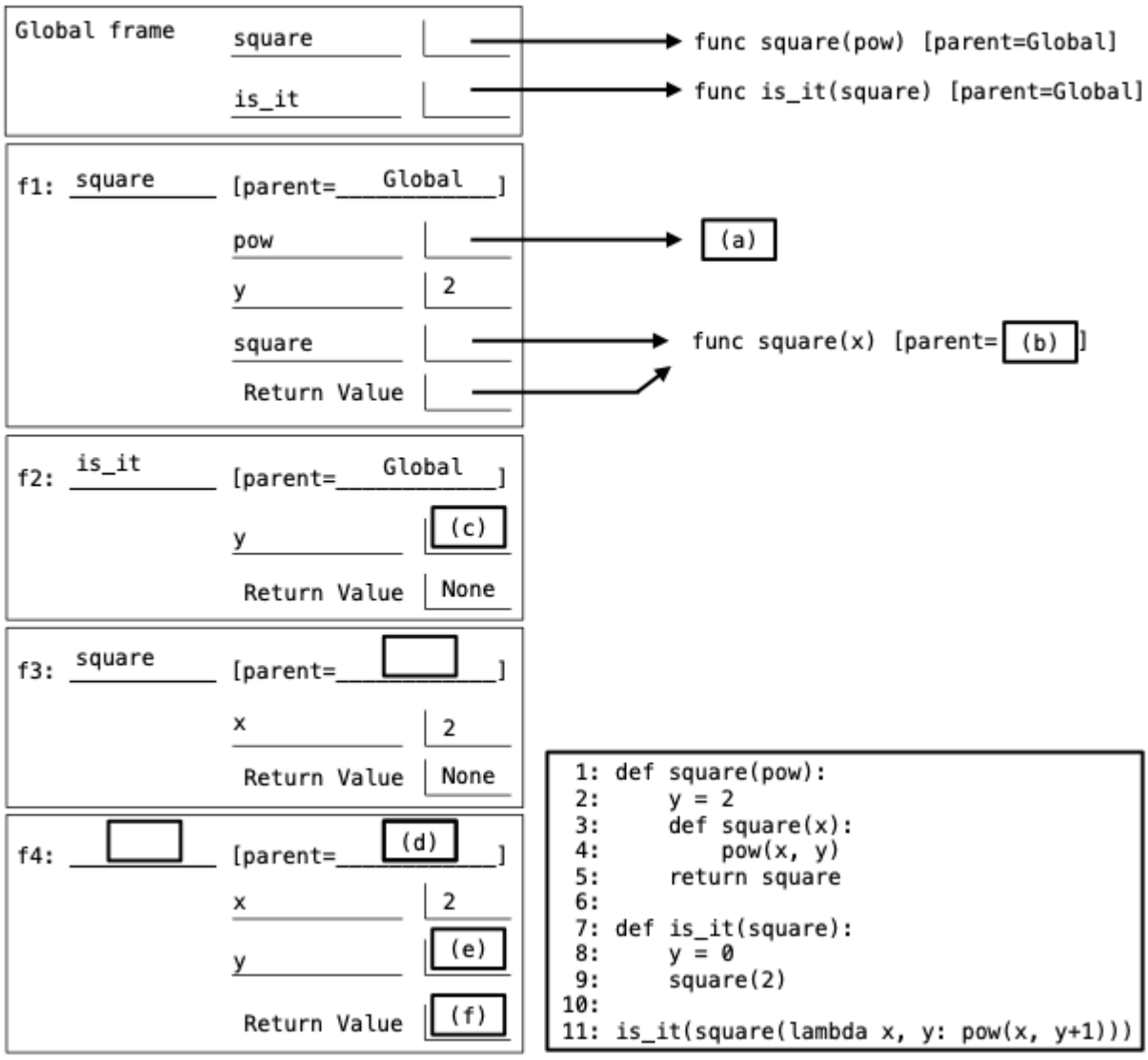
8
7

(d) (2.0 pt) `special(str, 10)`

'1010'

2. (6.0 points) Square the Square

Complete the environment diagram below and then answer the questions that follow. There is a one question for each labeled blank in the diagram. The blanks with no labels have no questions associated with them and are not scored. **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.**



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

- ○ `func pow(...) [parent=Global]`
- ○ `func pow(...) [parent=f1]`
- ● `func lambda <line 11>(x, y) [parent=Global]`
- ○ `func lambda <line 11>(x, y) [parent=f1]`

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

- [x] `f1`
- [ ] `Global`
- [ ] `square`
- [ ] None of these

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

- [x] `0`
- [ ] `1`
- [ ] `2`
- [ ] None of these

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

- [ ] `f1`
- [x] `Global`
- [ ] `square`
- [ ] None of these

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

- [ ] `0`
- [ ] `1`
- [x] `2`
- [ ] `3`
- [ ] None of these

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

- [ ] `1`
- [ ] `2`
- [ ] `4`
- [x] `8`
- [ ] None of these

3. (14.0 points) On Repeat

Definition: A positive integer `n` is a *repeating sequence* of positive integer `m` if `n` is written by repeating the digits of `m` one or more times. For example, 616161 is a repeating sequence of 61, but 61616 is not.

Hint: `pow(b, n)` raises `b` to the power of `n`. For example, `pow(10, 3)` is `1000`, and `654321 % pow(10, 3)` is `321` (the last 3 digits).

(a) (8.0 points)

Implement `repeating` which takes positive integers `t` and `n`. It returns whether `n` is a repeating sequence of some `t`-digit integer. **You may not use `str` or `repr` or `[` or `]` or `for`.**

```
def repeating(t, n):
    """Return whether t digits repeat to form positive integer n.

    >>> repeating(1, 6161)
    False
    >>> repeating(2, 6161)  # repeats 61 (2 digits)
    True
    >>> repeating(3, 6161)
    False
    >>> repeating(4, 6161)  # repeats 6161 (4 digits)
    True
    >>> repeating(5, 6161)  # there are only 4 digits
    False
    """
    if _________ > n:  # check that n has at least t digits
          (a)

        return False

    rest = n

    while rest:

        if rest % pow(10, t) != _________:
                                    (b)

            _________
               (c)


        _________
           (d)

    return _________
              (e)
```

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

- ○ `t`
- ○ `t - 1`
- ○ `10 * t`
- ○ `10 * (t - 1)`
- ○ `pow(10, t)`
- ● `pow(10, t - 1)`

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

```
n % pow(10, t)
```

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

```
return False
```

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

```
rest = rest // pow(10, t)
```

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

- (●) `True`
- ( ) `False`
- ( ) `rest == n`
- ( ) `rest != n`

(b) (6.0 points)

Implement `digits`, which takes a positive integer `n`. It returns the smallest `m` for which `n` is a repeating sequence of `m`. **You may not use `str` or `repr` or `[` or `]` or `for`.** Assume `repeating` is implemented correctly (and you may use it).

```
def digits(n):
    """Return the smallest m that repeats to form n.

    >>> digits(6161616161)
    61
    >>> digits(33333333)
    3
    >>> digits(12312312)
    12312312
    >>> digits(123123123)
    123
    """
    k = 1

    while True:

        if _________:
              (f)

            return _________
                       (g)
        _________
           (h)
```

i. (3.0 pt) Fill in blank (f).

```
repeating(k, n)
```

ii. (2.0 pt) Fill in blank (g).

```
n % pow(10, k)
```

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

- (●) `k = k + 1`
- ( ) `n = n // 10`
- ( ) `return False`
- ( ) `return n`

4. (10.0 points) Perfect Ten

Definitions:

- A *non-negative function* takes one argument and returns a non-negative integer.
- The *nearest* multiple of 10 to a non-negative integer `n` is larger than `n` if the last digit of `n` is 5 or more and less than or equal to `n` otherwise. For example, the nearest multiple of 10 to 25 is 30, but the nearest multiple of 10 to 24 is 20.

(a) (5.0 points)

Implement `nearest_ten`, which takes a non-negative function `f`. The function returned by calling `nearest_ten(f)` takes one argument `n` and returns the nearest multiple of 10 to `f(n)`. (In other words, it rounds `f(n)` to the nearest ten.)

```
def nearest_ten(f):
    """nearest_ten returns a function of n that returns the multiple of 10 nearest to f(n).

    >>> h = nearest_ten(lambda n: n * n)
    >>> h(2)  # The nearest multiple of 10 to 2 * 2 = 4
    0
    >>> h(3)  # The nearest multiple of 10 to 3 * 3 = 9
    10
    >>> h(5)  # The nearest multiple of 10 to 5 * 5 = 25
    30
    >>> h(8)  # The nearest multiple of 10 to 8 * 8 = 64
    60
    >>> h(10)  # The nearest multiple of 10 to 10 * 10 = 100
    100
    """
    def g(n):

        y = _________
              (a)

        rest, ones = y // 10, y % 10

        if _________:
             (b)

            rest = _________
                      (c)

        return _________
                  (d)

    return g
```

i. (2.0 pt) Fill in blank (a). **You may not use `round` or `int` or `//` for this blank.**

`f(n)`

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

- ○ `rest > 5`
- ○ `rest >= 5`
- ○ `ones > 5`
- ● `ones >= 5`
- ○ `y > 5`
- ○ `y >= 5`

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

- ○ `rest + ones`
- ○ `rest * 10 + ones`
- ○ `rest // 10`
- ○ `rest % 10`
- ● `rest + 1`

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

- ○ `rest`
- ● `rest * 10`
- ○ `rest + ones`
- ○ `rest * 10 + ones`
- ○ `rest + round(ones)`
- ○ `rest * 10 + round(ones)`
- ○ `round(rest + ones)`
- ○ `round(rest * 10 + ones)`

(b) (5.0 points)

Implement `distance`, which takes two arguments: a positive integer function `f` and a positive integer `n`. It returns the absolute difference (a non-negative integer) between `f(n)` and the nearest multiple of 10 to `f(n)`. **You may not call `round` or `int` or use `+` or `-`.** Assume `nearest_ten` is implemented correctly (and you may use it).

```
def distance(f, n):
    """Return the absolute value of the difference between f(n)
    and the nearest multiple of 10 to f(n).

    >>> distance(lambda n: n * n, 4)  # 16 is 4 away from 20
    4
    >>> distance(lambda n: n * n, 9)  # 81 is 1 away from 80
    1
    """
    return abs(_________ - _________)
                  (e)         (f)
```

i. (2.0 pt) Fill in blank (e).

- ○ `f`
- ● `f(n)`
- ○ `n`
- ○ `n * n`
- ○ `lambda n: n`
- ○ `lambda n: f(n)`
- ○ `lambda n: n * n`
- ○ `(lambda n: n * n)(n)`

ii. (3.0 pt) Fill in blank (f).

```
nearest_ten(f)(n)
```

(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!

Use `nearest_ten` to implement `nearest_hundred`, which takes a positive integer function `f`. The function returned by calling `nearest_hundred(f)` takes a positive integer `n` and returns the nearest multiple of 100 to `f(n)`. **Note:** The nearest multiple of 100 to `n` is larger than `n` if and only if the tens digit of `n` is 5 or more, so 300 is nearest to 250, but 200 is nearest to 249. **You may not call `round` or `int`. You may not use `str` or `repr` or `%`.** Assume `nearest_ten` is implemented correctly, and you **must** use it.

```
def nearest_hundred(f):
    """nearest_hundred(f)(n) is the multiple of 100 nearest to f(n).

    >>> h = nearest_hundred(lambda n: n * n)
    >>> h(8)  # The nearest multiple of 100 to 8 * 8 = 64
    100
    >>> h(20)  # The nearest multiple of 100 to 20 * 20 = 400
    400
    >>> h(25)  # The nearest multiple of 100 to 25 * 25 = 625
    600
    """
    return _________
```

i. Fill in the blank.

```
lambda n: nearest_ten(lambda y: f(y) // 10)(n) * 10
lambda n: nearest_ten(lambda y: y // 10)(f(n)) * 10
lambda n: nearest_ten(lambda y: y)(f(n) // 10) * 10
```

5. Just for Fun

(a) Optional: If CS 61A were a sport, what would it look like? (Feel free to draw.)