CS 61A Midterm 1

Ece Tabag

TOTAL POINTS

28.5 / 40

QUESTION 1

What Would Python Display? 10 pts

1.1 (a) print(0, print(1), 2) 2 / 2

✓ **+ 1 pts** *`1` appears on its own line as the first line*

✓ **+ 1 pts** *`0 None 2` appears anywhere on its own line. Other variants accepted:*

`0None2` `0,None,2` `0, None, 2`

+ 0.5 pts [partial] `1` appears on its own line (but not as the first line) or at the start of a longer line

+ 0.5 pts [partial] `0 None 2` appears, but as part of a line instead of on its own line

+ 0.5 pts [partial] There are three separate lines of `0` then `None` then `2` in that order (but not necessarily consecutive).

+ 0 pts [no penalty] extra `None` at the end

+ 0 pts Blank / no credit

1.2 (b) print(glob(e), glob(3), e) 3 / 4

- 0 pts all numbers are correct

```
40
3
41 4 40
```

commas allowed, final `None` allowed

For remote exams, the middle value `4` can be any integer `2` through `10`. The integer must be exactly one more than the integer on the previous line.

✓ **- 1 pts** *one number is wrong/missing*

- 2 pts two numbers are wrong/missing

- 3 pts three numbers are wrong/missing

- 0.5 pts new line error(s)

- 0.5 pts incorrect order

- 0.5 pts extra output (not including the final `None`) or extra quotes

- 4 pts incorrect / blank

1.3 (c) mid(not 'swift') 2 / 2

✓ **+ 1 pts** *First line: 8 or starts with 8*

✓ **+ 1 pts** *Last line: 7 or ends with 7*

- 0.5 pts Extra incorrect lines

- 0.5 pts Incorrect output formatting, e.g. `'7'`

+ 0 pts No new line

+ 0 pts Blank / no credit

1.4 (d) special(str, 10) 1 / 2

+ 2 pts `'1010'` or `"1010"` (but not `"10 10"`)

For remote exams, `10` can be any two-digit integer.

✓ **+ 1 pts** *Answer is or contains `1010` or `10 10`. (e.g "10 10")*

Not accepted:

`'10' '10'`

`str(10) + str(10)`

`10 + 10`

`'10' + '10'`

For remote exams, `10` can be any two-digit integer.

+ 0 pts Blank / no credit

QUESTION 2

Square the Square 6 pts

2.1 (a) 1 / 1

+ 0 pts func pow(...) [parent=Global]

+ 0 pts func pow(...) [parent=f1]

✓ **+ 1 pts** *func lambda <line 11>(x, y) [parent=Global]*

+ 0 pts func lambda <line 11>(x, y) [parent=f1]

+ 0 pts Blank / no credit

2.2 (b) 1 / 1

✓ **+ 1 pts** *f1*

+ 0 pts Global

+ 0 pts square

+ 0 pts None of these

+ 0 pts Blank

2.3 (c) 1 / 1

✓ **+ 1 pts** *0*

+ 0 pts 1

+ 0 pts 2

+ 0 pts None of these

2.4 (d) 0 / 1

✓ **+ 0 pts** *f1*

+ 1 pts Global

+ 0 pts square

+ 0 pts None of these

+ 0 pts Blank

2.5 (e) 0 / 1

+ 1 pts 2

+ 0 pts 0

+ 0 pts 1

+ 0 pts 3

✓ **+ 0 pts** *None of these*

+ 0 pts Blank

2.6 (f) 1 / 1

✓ **+ 1 pts** *8*

+ 0 pts 1

+ 0 pts 2

+ 0 pts 4

+ 0 pts None of these

+ 0 pts Blank

QUESTION 3

On Repeat 14 pts

3.1 repeating (a) 2 / 2

✓ **+ 2 pts** *pow(10, t - 1)*

+ 0 pts pow(10, t)

+ 0 pts t

+ 0 pts t - 1

+ 0 pts 10 * (t - 1)

+ 0 pts 10 * t

+ **0 pts** Blank

3.2 repeating (b) **0 / 2**

+ **2 pts** Check against the end: `n % pow(10, t)`

`t` can be `k`, `d`, or `m`

+ **1 pts** [partial] Check against the middle:
`rest % pow(10, 2*t) // pow(10, t)`
`rest // pow(10, t) % pow(10, t)`

`t` can be `k`, `d`, or `m`

+ **1 pts** [partial] Incorrect way to check against the end:
`n // pow(10,t)`
`n / pow(10,t)`
`n % pow(10,_)` for t-1, t+1

`t` can be `k`, `d`, or `m`

✓ **+ 0 pts** *Blank / no credit*

Including (but not limited to):
`rest % pow(10, __)`
`rest // pow(10, __)`
`rest`
`n`

3.3 repeating (c) **1 / 1**

✓ **+ 1 pts** *`return False`*

+ **1 pts** `return` using an expression that always evaluates to `False` when reached. Examples:

`rest % pow(10, t) != n % pow(10, t)`

+ **0.5 pts** `False` or `false` or an expression that always evaluates to `False` when reached (with no `return`)

+ **0 pts** Blank / no credit

3.4 repeating (d) **0.5 / 2**

+ **2 pts** `rest = rest // pow(10, t)`
`rest //= pow(10, t)`

`t` can be `k`, `d`, or `m`

+ **1.5 pts** [partial] `_ = rest // pow(10, t)`
e.g. `n = rest // pow (10, t)`

`t` can be `k`, `d`, or `m`

+ **1 pts** [partial] `rest // pow(10, t)` without reassignment
or `rest = rest // pow(10,t)` along with some other unnecessary code

`t` can be `k`, `d`, or `m`

+ **1 pts** [partial] `rest = rest _ pow(10, _)` E.g.,
`rest = rest % pow(10, t)`
`rest = rest // pow(10, t-1)`
`rest = rest / pow(10, t)`

`t` can be `k`, `d`, or `m`

+ **1 pts** `n = n//pow(10, t)`

`t` can be `k`, `d`, or `m`

✓ **+ 0.5 pts** *[partial] `rest = rest // 10` or `rest = rest // (10*t)`*

`t` can be `k`, `d`, or `m`

+ **0 pts** Blank / no credit

3.5 repeating (e) 1 / 1

✓ **+ 1 pts** *True*

+ 0 pts False

+ 0 pts rest == n

+ 0 pts rest != n

+ 0 pts Blank

3.6 digits (f) 0 / 3

+ 3 pts `repeating(k, n)`
`repeating(k, n) == True`

OK if this appears as one clause in a logical expression using `and`/`or` [see rubric items 4 and 5].

`k` can be `count`, `num`, or `a`

+ 2 pts [partial] `repeating(_, n)` where the first blank involves `k`. E.g.,

`repeating(k+1, n)`

OK if this appears as one clause in a logical expression using `and`/`or` [see rubric items 4 and 5].

`k` can be `count`, `num`, or `a`

+ 1 pts [partial] Any call to `repeating` that does not meet one of the above rubric items

✓ **+ 0 pts** *[no penalty] `and`/`or` clause combined with a correct/partially correct call to `repeating` that still makes the answer correct/partially correct. Examples:*

...

- 1 pts call to wrong function (ex. returning instead of repeating)
swapped parameters (ex. repeating(n, k))

`and`/`or` clause combined with a correct/partially correct call to `repeating` that makes the answer incorrect.

repeating(k, n) == False

...

- 0.5 pts Minor syntax error
(e.g., use single `=` instead of `==`)
writing 'True' instead of True
using `t` instead of `k` or `m` instead `n`

+ 0 pts Blank / no credit

3.7 digits (g) 2 / 2

✓ **+ 2 pts** *`n % pow(10, k)`*

OK to write `return` first

`k` can be `count`, `num`, or `a`

+ 1 pts [partial] `n _ pow(10, _)` E.g.,
`n // pow(10, k)`
`n % pow(10, k-1)`

`k` can be `count`, `num`, or `a`

+ 0.5 pts `k`

`k` can be `count`, `num`, or `a`

+ 0 pts Blank / no credit

3.8 digits (h) 1 / 1

✓ **+ 1 pts** *k = k + 1*

+ 0 pts n = n // 10

+ 0 pts return False

+ 0 pts return n

+ 0 pts Blank

+ 0 pts Incorrect

QUESTION 4

Perfect Ten 10 pts

4.1 nearest_ten (a) 2 / 2

✓ **+ 2 pts** `*f(n)*`

`*abs(f(n))*`

`*y = f(n)*`

+ 1 pts `f`

`lambda n: f(n)`

+ 1 pts Just used squaring instead of `f`:

`n * n`

`(lambda n: n * n)(n)`

`pow(n, 2)`

But not

`lambda n: n * n`

+ 0 pts Blank / no credit

Including:

`n`

4.2 nearest_ten (b) 1 / 1

+ 0 pts rest > 5

+ 0 pts rest >= 5

+ 0 pts ones > 5

✓ **+ 1 pts** *ones >= 5*

+ 0 pts y > 5

+ 0 pts y >= 5

+ 0 pts Blank

4.3 nearest_ten (c) 0 / 1

+ 0 pts rest + ones

✓ **+ 0 pts** *rest * 10 + ones*

+ 0 pts rest // 10

+ 0 pts rest % 10

+ 1 pts rest + 1

+ 0 pts Blank

4.4 nearest_ten (d) 1 / 1

+ 0 pts rest

✓ **+ 1 pts** *rest * 10*

+ 0 pts rest + ones

+ 0 pts rest * 10 + ones

+ 0 pts rest + round(ones)

+ 0 pts rest * 10 + round (ones)

+ 0 pts round(rest + ones)

+ 0 pts round(rest * 10 + ones)

+ 0 pts Blank

4.5 distance (e) 2 / 2

+ 0 pts f

✓ **+ 2 pts** *f(n)*

+ 0 pts n

+ 0 pts n * n

+ 0 pts lambda n: n

+ 0 pts lamda n: f(n)

+ 0 pts lambda n: n * n

+ 0 pts (lambda n: n * n)(n)

+ 0 pts Blank

4.6 distance (f) 3 / 3

✓ **+ 3 pts** `*nearest_ten(f)(n)*`

`*nearest_ten(lambda n: f(n))(n)*`

`*nearest_ten(lambda n: n)(f(n))*`

+ 2 pts `lambda _: nearest_ten(f)(n)` where _ is any variable;
`nearest_ten(f)(n) [some extra work]`

+ 1 pts `nearest_ten(f(n))`
`nearest_ten(f, n)`

+ 1 pts [partial] called `nearest_ten` on a function. E.g.,
`nearest_ten(f)`

+ 0.5 pts [partial] Calls `nearest_ten` on anything
`nearest_ten(n)`
`nearest_ten(n*n)`

+ 0 pts Blank / no credit

4.7 A+: nearest_ten **0 / 0**

+ 0 pts Correct. Irrelevant syntax errors (e.g., `x` instead of `*`) are ok as long as the logic and function calling is perfect.

`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`

✓ **+ 0 pts** *Attempted but incorrect*

+ 0 pts Blank

QUESTION 5

Just for Fun 0 pts

5.1 Drawing **0 / 0**

✓ **+ 0 pts** *Not blank*

+ 0 pts Blank

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?

Ece Tabag

(b) What is your student ID number?

3037995612

(c) What is your @berkeley.edu email address?

ecetabag @ berkeley.edu

(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.

ET

1. (10.0 points) What Would Python Display?

Assume the following code has been executed.

```
c = 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 3 40
```

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

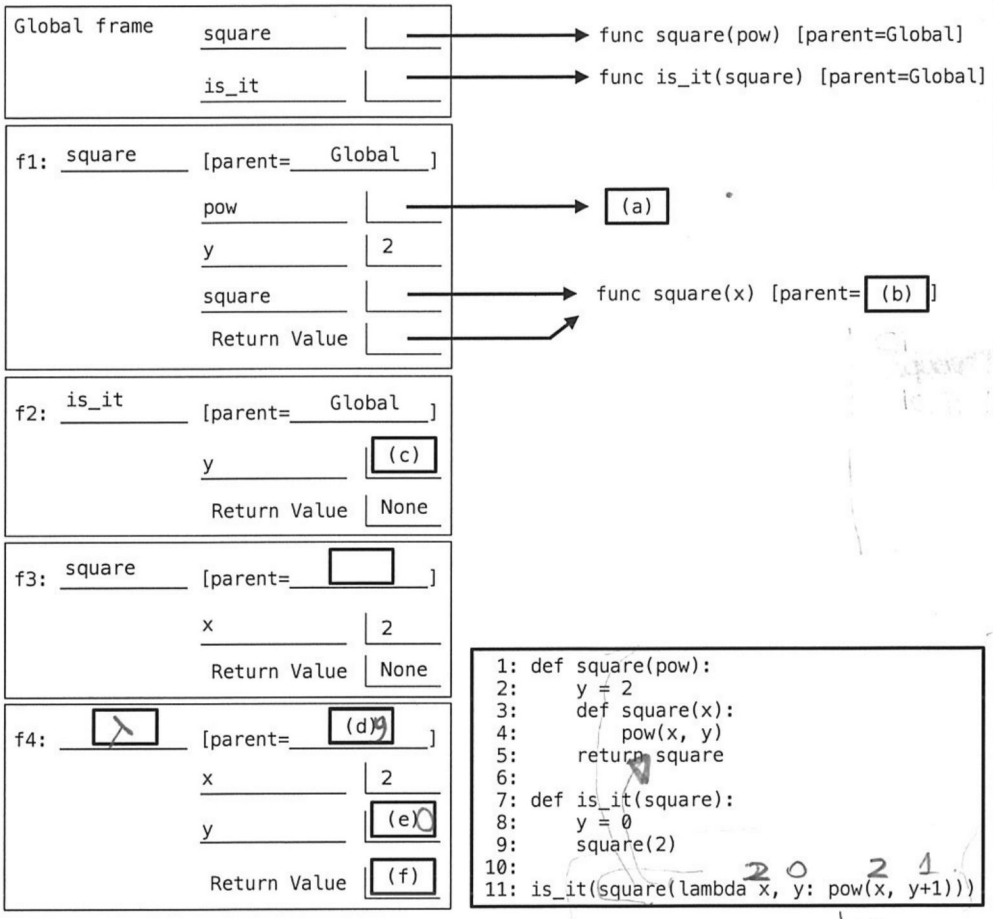
```
8
7
```

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

```
10 10
```

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.**




```
 1: def square(pow):
 2:     y = 2
 3:     def square(x):
 4:         pow(x, y)
 5:     return square
 6:
 7: def is_it(square):
 8:     y = 0
 9:     square(2)
10:
11: is_it(square(lambda x, y: pow(x, y+1)))
```

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

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

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

- ● `f1`
- ○ `Global`
- ○ `square`
- ○ None of these

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

- (●) `0`
- ( ) `1`
- ( ) `2`
- ( ) None of these

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

- (●) `f1`
- ( ) `Global`
- ( ) `square`
- ( ) None of these

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

- ( ) `0`
- ( ) `1`
- ( ) `2`
- ( ) `3`
- (●) None of these

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

- ( ) `1`
- ( ) `2`
- ( ) `4`
- (●) `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 pow(10, t-1) > n:  # check that n has at least t digits
          (a)
        return False
    rest = n
    while rest:
        if rest % pow(10, t) != rest // pow(10,t):
                                  (b)
            return False
                   (c)
        rest = rest // 10
                   (d)
    return True
                (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)

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

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

```
return False
```

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

```
rest = rest//10
```

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).

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

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
```

f(n) = 64

4 | 10

2 5

ones >= 5

rest * 10 + ones

rest * 10

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)`

ab f

difference

f(n) - nearest_10(f)(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.

5. Just for Fun

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