INSTRUCTIONS

• You have 2 hours to complete the exam.

• The exam is closed book, closed notes, closed computer, closed calculator, except one hand-written 8.5" × 11" crib sheet of your own creation and the official CS 61A midterm 1 study guide.

• Mark your answers on the exam itself. We will not grade answers written on scratch paper.

<table>
<thead>
<tr>
<th>Last name</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>First name</td>
<td></td>
</tr>
<tr>
<td>Student ID number</td>
<td></td>
</tr>
<tr>
<td>CalCentral email (<a href="mailto:_@berkeley.edu">_@berkeley.edu</a>)</td>
<td></td>
</tr>
<tr>
<td>TA</td>
<td></td>
</tr>
<tr>
<td>Name of the person to your left</td>
<td></td>
</tr>
<tr>
<td>Name of the person to your right</td>
<td></td>
</tr>
</tbody>
</table>

Policies & clarifications

• 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 min, max, pow, and abs. You may not use functions defined on your study guide unless clearly specified in the question.

• For fill-in-the blank coding problems, we will only grade work written in the provided blanks. You may only write one Python statement per blank line, and it must be indented to the level that the blank is indented.

• Unless otherwise specified, you are allowed to reference functions defined in previous parts of the same question.
1. (12 points) Frame of Thrones

For each of the expressions in the table 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”. The first two rows have been provided as examples.

The interactive interpreter displays the value of a successfully evaluated expression, unless it is None.

Assume that you have first started python3 and executed the statements on the left.

```
from operator import add, sub

def winterfell(a, b):
    a
    b
    return b(a+1, b(a))

da, ny = 20, 18

while da > ny:
    da = ny
    da, ny = ny + 1, da + 3

def tar(gar, yen):
    if print(yen):
        print(yen + 1)
    return gar(yen)

def st(ar, k=None):
    return lambda a, y: ar(y, a)

night = st(sub)
king = st(st(pow))

def jon(sn, ow):
    print(ow)
    jon = sn(ow)
    print(ow)
    return jon

def snow(ow):
    def tarly(snow):
        return ow + snow
    ow += 2
    return tarly
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub(pow(10, 2), 1)</td>
<td>99</td>
</tr>
<tr>
<td>print(4, 5) + 1</td>
<td>4 5 Error</td>
</tr>
<tr>
<td>(print(2) or 3) // (0 or 1)</td>
<td>2 3</td>
</tr>
<tr>
<td>winterfell(2, print)</td>
<td>2 3 None</td>
</tr>
<tr>
<td>ny</td>
<td>21</td>
</tr>
<tr>
<td>tar(lambda x: x-7, 8)</td>
<td>8 1</td>
</tr>
<tr>
<td>night(king(2, 3), 4)</td>
<td>-4</td>
</tr>
<tr>
<td>jon(snow(5), 2)</td>
<td>2 2 9</td>
</tr>
</tbody>
</table>
2. (8 points) Stranger Frames

Fill in the environment diagram that results from executing the code on the right until the entire program is finished, an error occurs, or all frames are filled. You may not need to use all of the spaces or frames. A complete answer will:

- Add all missing names and parent annotations to all local frames.
- Add all missing values created or referenced during execution.
- Show the return value for each local frame.

```python
# Global frame
lucas         
dustin        
will 11

# f1: dustin [parent= Global]
lucas         
will 1

dustin        

# f2: λ [parent= f1]
mad           

# f3: dustin [parent= f1]
mike 3 
will 2

# f4: lucas [parent= Global]
mike 1

# Code execution

def lucas(mike):
    return will

def dustin(lucas):
    will = 1
    def dustin(mike):
        will = 2
        return lucas
    return lambda mad: dustin(3)(will)

will = 5 + 6
lucas = dustin(lucas)
lucas(max)
```

The environment diagram would show the execution flow and values as follows:

- The global frame includes the variables `lucas`, `dustin`, and `will`.
- The `f1` frame is created due to the `dustin` call, showing that `will` is initialized to 1.
- The `f2` frame is created due to the lambda call, with `mad` as the variable.
- The `f3` frame is created due to the `dustin(lucas)` call, showing that `mike` and `will` are defined.
- The `f4` frame is created due to the `lucas` call, showing that `mike` is defined.

The final value of `lucas(max)` would be shown as 11.
3. (10 points) Choose Wisely

(a) (4 pt) Implement sum_some, which takes a non-negative integer \( n \) and a function \( p \). It returns the sum of all the digits \( d \) for which \( p \) returns a true value when given \( d \) as an argument. Assume that the function \( p \) takes a single digit \( d \) (from 0 to 9) and returns either True or False.

```python
def sum_some(n, p):
    """Return the sum of the digits of \( N \) for which \( P \) returns a true value."

    >>> even = lambda d: d % 2 == 0
    >>> big = lambda d: d > 5
    >>> sum_some(124567, even)  # Sum the even digits: 2 + 4 + 6
    12
    >>> sum_some(124567, big)   # Sum the big digits: 6 + 7
    13
    """
    total = 0
    while n:
        if p(n % 10):
            total += n % 10
        n = n // 10
    return total
```

(b) (4 pt) Implement sum_largest, which takes non-negative integers \( n \) & \( k \). It sums the largest \( k \) digits of \( n \).

```python
def sum_largest(n, k):
    """Return the sum of the \( K \) largest digits of \( N \)."

    >>> sum_largest(2018, 2)  # 2 and 8 are the two largest digits (larger than 0 and 1).
    10
    >>> sum_largest(12345, 10) # There are only five digits, so all are included in the sum.
    15
    """
    if n == 0 or k == 0:
        return 0
    a = n % 10 + sum_largest(n // 10, k - 1)
    b = sum_largest(n // 10, k)
    return max(a, b)
```

(c) (2 pt) Complete the expression below by only adding parentheses so that the whole expression evaluates to 2018. Each blank should be filled with one or more parentheses.

```
(lambda a, x: x + (lambda y: lambda z: y+z+1000)(1000 _)(_ 10 _)(_ 5, _(_ lambda: 8 _)())_ )
```
4. (10 points) Editor

Definitions. An edit is a pure function that takes a non-negative integer and returns a non-negative integer. An editor for a non-negative integer n is a function that takes an edit, applies it to n, displays the result, and then returns an editor for the result.

(a) (3 pt) Implement make_editor, which takes a non-negative integer n and a one-argument function pr. It returns an editor for n that uses pr to display the result of each edit.

(b) (5 pt) Implement insert, which takes a single digit d (from 0 to 9) and a non-negative position k. It returns an edit that inserts d into its argument n at position k, where k counts the number of digits from the end of n. Assume that k is not larger than the number of digits in n. Your solution must be recursive.

(c) (2 pt) Implement delete, which takes a non-negative integer k and returns an edit that deletes the last k digits of its argument n. You may use pow or ** in your solution.

```python
def make_editor(n, pr):
    """Return an editor for N."""
    editor = lambda edit: make_editor(edit(n), pr)
    return editor

def insert(d, k):
    def edit(n):
        if k == 0:
            return d + 10 * n
        else:
            return n % 10 + 10 * insert(d, k-1)(n//10)
    return edit

delete = lambda k: lambda n: n // 10 ** k
```

```python
>>> f = make_editor(2018, lambda n: print('n is now', n))
>>> f = f(delete(3)) # delete the last 3 digits from the end of 2018
n is now 2
>>> f = f(insert(4, 0)) # insert digit 4 at the end of 2 (position 0)
n is now 24
>>> f = f(insert(3, 1)) # insert digit 3 in the middle of 24 (position 1)
n is now 234
>>> f = f(insert(1, 3)) # insert digit 1 at the start of 234 (position 3)
n is now 1234
>>> f = make_editor(123, print)(delete(10)) # delete 10 digits from the end of 123
0
"""
No more questions.