PYTHON, FUNCTIONS, EXPRESSIONS, AND CONTROL Meta

COMPUTER SCIENCE MENTORS 61A

January 23–January 27, 2023

Recommended Timeline

- Introductions/Expectations/Icebreaker [10 minutes]
- Q1: What Would Python Display + Minilecture/explanations [15 minutes]
- Q2: Order of evaluation [5 min]

  Note: don't need to do all parts if students get the idea
- Code Writing [20 minutes]

  - Again, no need to do all questions

  - 2 and 3 are medium difficulty & the last one is a bit more challenging, pick based on how your students are feeling
- Tips and Tricks for succeeding in CS61A [leftover time]

1 Intro to Python

1. What Would Python Display?

```
>>> 3

3

>>> "cs61a"

'cs61a'

>>> x = 3
>>> x

3

>>> x = print("cs61a")
cs61a
>>> x
```

Created by Alyssa Smith, Esther Shen

```
None

>>> print(print(print("cs61a")))


cs61a
None
None

>>> def f1(x):
...     return x + 1
>>> f1(3)

4

>>> f1(2) + f1(2 + 3)

9

>>> def f2(y):
...     return y / 0
>>> f2(4)

ZeroDivisionError: division by zero

>>> def f3(x, y):
...     if x > y:
...             return x
...     elif x == y:
...             return x + y
...     else:
...             return y
>>> f3(1, 2)

2

>>> f3(5, 5)

10

>>> 1 or 2 or 3

1

>>> 1 or 0 or 3

1

>>> 4 and (2 or 1/0)

2

>>> 0 or (not 1 and 3)
```

```
False
>>> (2 or 1/0) and (False or (True and (0 or 1)))

1
```

2. For the following expressions, simplify the operands in the order of evaluation of the entire expression

   Example: `add(3, mul(4, 5))`

   Order of Evaluation: `add(3, mul(4, 5))` → `add(3, 20)` → `23`

   (a) `add(1, mul(2, 3))`

   ```
   add(1, mul(2, 3))
   add(1, 6)
   7
   ```

   (b) `add(mul(2, 3), add(1, 4))`

   ```
   add(mul(2, 3), add(1, 4))
   add(6, add(1, 4))
   add(6, 5)
   11
   ```

   (c) `max(mul(1, 2), add(5, 6), 3, mul(mul(3, 4), 1), 7)`

   ```
   max(mul(1, 2), add(5, 6), 3, mul(mul(3, 4), 1), 7)
   max(mul(1, 2), add(5, 6), 3, mul(12, 1), 7)
   max(2, add(5, 6), 3, mul(12, 1), 7)
   max(2, 11, 3, mul(12, 1), 7)
   max(2, 11, 3, 12, 7)
   12
   ```

   It's probably not necessary to get through all the parts of this problem if your students get the idea. The last subpart is probably the most instructive. For question 2, feel free to remind students of the general "order of operations" of functions, in that they start inward and expand outward. Feel free to use any analogies, that help students understand. If your students are still confused, it's advisable to walk through the first few problems with them step by step.

2 Control

1. Write a function that returns `True` if a number is divisible by 4 and `False` otherwise.

```
def is_divisible_by_4(num):
    return num % 4 == 0
```

The purpose of this problem is essentially to ensure that students are familiar with Python syntax. Its solution involves little complex thought.

2. Implement `fizzbuzz(n)`, which prints numbers from 1 to `n` (inclusive). However, for numbers divisible by 3, print "fizz". For numbers divisible by 5, print "buzz". For numbers divisible by both 3 and 5, print "fizzbuzz".

```
def fizzbuzz(n):
    """
    >>> result = fizzbuzz(16)
    1
    2
    fizz
    4
    buzz
    fizz
    7
    8
    fizz
    buzz
    11
    fizz
    13
    14
    fizzbuzz
    16
    >>> result is None
    True
    """


    i = 1
    while i <= n:
        if i % 3 == 0 and i % 5 == 0:
            print('fizzbuzz')
        elif i % 3 == 0:
            print('fizz')
        elif i % 5 == 0:
            print('buzz')
        else:
            print(i)
        i += 1
```

We must put the condition `i % 3 == 0 and i % 5 == 0` in the first **`if`**. For example, if we were to write the body of the while loop with the first two conditions switched.

```
        if i % 3 == 0:
            print('fizz')
        elif i % 3 == 0 and i % 5 == 0:
            print('fizzbuzz')
        elif i % 5 == 0:
            print('buzz')
        else:
            print(i)
```

then we may print out 'fizz' for a number like 15 which would be incorrect.

Students may overthink this problem, trying to find a "clever" way to do it with a smaller number of comparisons. There are many variations of this problem. If students solve this easily, encourage them to find a way to solve this with a for loop, or some other variation.

3. Implement `pow_of_two`, which prints all the positive integer powers of two less than or equal to `n` in ascending order. This function should return `None`.

   *Follow up question: What would you change about your solution if the question asked to print all the powers of two **strictly less than** `n`?*

```
def pow_of_two(n):
    """
    >>> pow_of_two(6)
    1
    2
    4
    >>> result = pow_of_two(16)
    1
    2
    4
    8
    16
    >>> result is None
    True
    """


    curr = 1
    while curr <= n:
        print(curr)
        curr *= 2 # equivalent to curr = curr * 2
```

Since we are multiplying `curr` by 2 on each iteration of the while loop, `curr` holds values that are powers of 2. Notice that since there is no return statement in this function, when Python reaches the end of the function, it automatically returns `None`.

The answer to the follow up question is that the condition of our while loop would change to `curr < n`. Walk through the code for `pow_of_two(16)` with both of the conditions to see why they produce different outputs!

Another way you could have written this function is by using **pow** or the `**` operator. That solution would look something like this where you would keep track of the exponent itself:

```
    exponent = 0
    while (2 ** exponent) <= n:
        print(2 ** exponent)
        exponent += 1
```

If your students are not familiar with factorials, you may want to give them a brief overview before going over this problem.

4. Complete the function `fact_limit`, which calculates factorials up to a specified limit. Specifically, `fact_limit` takes in two positive integers, `n` and `limit`, and calculates the product of `n`, `n-1`, `n-2`, etc., working downward until it attains the greatest product that doesn't exceed `limit`. If there is no product less than or equal to `limit`, `fact_limit` should return 1.

   Hint: The output of `fact_limit` is always less than or equal to `limit`.

```
def fact_limit(n, limit):
    """
    >>> fact_limit(5, 20)
    20 # 5 * 4 = 20, but 5 * 4 * 3 = 60 > 20
    >>> fact_limit(5, 200)
    120 # 5 * 4 * 3 * 2 * 1 = 120 < 200
    >>> fact_limit(5, 3)
    1 # no partial product is less than 3
    """
    if ________________:

        ________________

    product = ________________

    ________________ = n - 1

    while ________________:

        ____________ = ____________

        ____________ = ____________

    return ________________
```

```
def fact_limit(n, limit):
    if n > limit:
        return 1
    product = n
    n = n - 1
    while product * n <= limit and n > 0:
        product = product * n
        n = n - 1
    return product
```

This question differs significantly from a similar question that was used in past years. Please review it carefully before teaching it. One area in which students may need help is addressing the edge case in which n is greater than the limit.