Python Functions, Expressions & Control Meta

Computer Science Mentors 61A

January 27 – January 31, 2024

Example Timeline

- Intros, Ice Breakers, Expectations & Logstics [15 min]

  Don't be afraid to spend some time on this! It'll likely make your job in the future easier! This worksheet is relatively short.

- Mini-lecture + Q1: WWPD [10 min]
- Q2: Py function order of operations [5 min]

  Feel free to skip parts

- Coding [15-17 min]

  Note: **Each problem's "Suggested Time" is not assigned with the 1 hour time for sections in mind.** It is more of a note of how long each individual problem may take. That is, if you added all the "Suggested Times" in the coding section, it could very well be greater than 1 hour (especially in more complicated worksheets in the future).

  However, the time in brackets will add up to 50 minutes, and is intended to be a suggested outline for your section.

- How to be successful in 61A [3-5 min]

  Give any tips you want!

1 Intro to Python

1. What Would Python Display?

```
>>> 3

3

>>> "cs61a"

'cs61a'

>>> x = 3
>>> x
```

Created by Jim Fang, Bryan Adolfo Ventura Benitez, Rajoshi Basu Worksheet feedback: links.jimfang.dev/feedback

```
3
```

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

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

   Suggested Time: 7 min

   - Again, you can probably skip a lot of these as soon as your students get the idea.

     The last one probably has the most teaching potential.
   - Your students might be new to solving "compound" questions like the last one; Consider giving them tips on how to break down such problems.

1. Write a function that returns `True` if a number is divisible by 4, 1 if a number is divisible by 7 and is not already divisible by 4, and returns `False` if neither condition is fulfilled.

```
def divisibility_check(num):
```

```
def divisibility_check(num):
    if num % 4 == 0:
        return True
    elif num % 7 == 0:
        return 1
    else:
        return False
```

This also works as an alternate solution:

```
def divisibility_check(num):
    return True if num % 4 == 0 else 1 if num % 7 == 0 else False
```

Suggested Time: 5 min; Difficulty: Easy

- Strongly recommend going over the alternate solution since it tends to appear on exams.

2. 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
```

Suggested Time: 7-10 min; Difficulty: Medium

- This question has **`while`** loops! You may want to discuss the difference between **`for`** and **`while`** loops.
- Consider also comparing the syntax style of Python loops to Java for loops (Since AP CompSci A teaches in Java and many students' understanding of for-loops will be from the Java for-loop syntax). Tl;dr: Python's loops are closer to Java For-each loops than the traditional Java for loop.
- Also make sure your students understand that print returns None and that it is valid for a Python

function to not have a return statement.

3. Write a function, `is_leap_year`, that returns true if a number is a leap year and false otherwise. A *leap year* is a year that is divisible by 4 but not by 100, except for years divisible by 400, which are leap years.

```
def is_leap_year(year):
    """
    Returns whether ``year'' is a leap year.
    >>> is_leap_year(2002)
    False
    >>> is_leap_year(2004)
    True
    >>> is_leap_year(2000)
    True
    >>> is_leap_year(1900)
    False
    >>> is_leap_year(2100)
    False
    """
    return _____________
```

```
def is_leap_year(year):
    return (year % 4 == 0 and year % 100 != 0) or year % 400 == 0
```

Suggested Time: 5 min; Difficulty: Medium This question is similar to Divisibility Check (Q1), except it tests for students' knowledge of how "and" and "or" work. Only do this question if you decide to skip Divisibility Check, as the concepts it exercises are very similar.

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
```

Suggested Time: 10 min; Difficulty: Hard

- This problem is probably optional and it'd be better to spend the rest of the time talking about general strategies for success in 61A. Unless you have advanced students in your section, I suggest skipping this question.