Control, Environment Diagrams Solutions

CS 61A

June 25, 2021

1 Control

Control is the use of boolean expressions to prevent or allow a block of code to run based on a condition. We use `if` statements and `while` loops in conjunction with these boolean expressions depending on how we want the code to behave. `if/elif/else` blocks will execute the first block of code for which the condition is `True`, whereas `while` loops will repeatedly execute a block of code while the condition is `True`.

1. To handle discussion section overflow, TAs may direct students to a more empty section that is happening at the same time.

   Write a function that takes in the number of students in two sections and prints out what to do if either section exceeds 30 students.

```
def handle_overflow(s1, s2):
    """
    >>> handle_overflow(27, 15)
    No overflow
    >>> handle_overflow(35, 29)
    Move to Section 2: 1
    >>> handle_overflow(20, 32)
    Move to Section 1: 10
    >>> handle_overflow(35, 30)
    No space left in either section
    """

    if s1 <= 30 and s2 <= 30:
        print("No overflow")
    elif s2 > 30 and s1 < 30:
        print("Move to Section 1:" + str(30 - s1))
    elif s1 > 30 and s2 < 30:
        print("Move to Section 2:" + str(30 - s2))
    else:
        print("No space left in either section")
```

   Video walkthrough

2. Implement `pow_of_two`, which takes in an integer `n` and prints all the positive, integer powers of two less than or equal to n. 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
```

3. Fill out the function `min_fact` which calculates the product of consecutive positive numbers starting from `n` and working downwards until the first point at which the product becomes greater than `margin`. It should return -1 if there is no product that is greater than margin.

```
def min_fact(n, margin):
    """
    >>> min_fact(5, 20) # 5 * 4 * 3
    60
    >>> min_fact(5, 200)
    -1
    >>> min_fact(5, 0)
    5
    """
    total, ____________________ = n, ____________________
    while ____________________:
        _____________, _____________ = _____________, _____________
    if ____________________:
        ____________________
    return ____________________
```

```
def min_fact(n, margin):
    total, n = n, n - 1
    while total <= margin and n > 0:
        total, n = total * n, n - 1
    if total < margin:
        return -1
    return total
```

2 Environment Diagrams

An **environment diagram** is a model we use to keep track of all the variables that have been defined and the values they are bound to. We will be using this tool throughout the course to understand complex programs involving several different assignments and function calls.

1. When do we make a new frame in an environment diagram?

   We make a new frame in an environment diagram when calling a user-defined function, or when we are applying the operator to the operand(s). This occurs after both the operator and operand(s) are evaluated.

2. Draw the environment diagram that results from running the following code.

```
def swap(x, y):
    x, y = y, x
    return print("Swapped!", x, y)

x, y = 60, 1
a = swap(x, y)
swap(a, y)
```

https://tinyurl.com/y68m6qdj

3. Draw the environment diagram that results from running the following code.

```
def funny(joke):
    hoax = joke + 1
    return funny(hoax)

def sad(joke):
    hoax = joke - 1
    return hoax + hoax

funny, sad = sad, funny
result = funny(sad(2))
```

https://tinyurl.com/y5lc4fez

3 Challenge Control Problem

1. Fill out the function `digit_div` which returns an integer that contains in any order all the digits of `k` that divide `n` evenly. If no such digit of `k` exists, the function should return 0. Assume that both `n` and `k` are positive integers.

```
def digit_div (n, k):
    >>> digit_div(4, 1234567890)
    421
    >>> digit_div(4, 2323)
    22
    >>> digit_div(7, 2323)
    0
    """
    ____________________
    while ____________________________:
        curr_digit = k % 10
        if ________________________________________________:
            ____________________________________________
        ____________________________________________
    return _________________
```

```
def digit_div (n, k):
    digits = 0
    while k > 0:
        curr_digit = k % 10
        if curr_digit != 0 and n % curr_digit == 0:
            digits = digits * 10 + curr_digit
        k //= 10
    return digits
```

Notice that in the condition for the **`if`** statement, we must check if `curr_digit` is 0 before we divide `n` by `curr_digit` otherwise, we may try to divide `n` by 0 which would error! This line works because of short circuiting; if the first expression in an **`and`** expression is a false value, then the second expression is never evaluated.