Decision Seeds - William Vongphanith, Josiah Moltz, Cynthia Tenezaca
IntroCS pd6 sec10
Lab04 - Controlling the Flow
2021-03-19
time cost: 4-5 hrs

Lab 4

Logs

(Note: the decision trees were written AFTER the code was written)

HW11

I'm not breaking this up by function since I didn't have much to log and I was hoping that someone else could contribute

- In the beginning I used boxes for everything, including "forks in the road" because I didn't really know how to make decision trees.
- I had to erase and redraw my trees several times because they became too circuitous after I added many cases.
- I narrowed each function down to "functional cases" and "invalid input cases" which helped me organize and streamline the process.
- In class I learned that decision trees should use diamonds to indicate "forks in the road".
- I remade the decision trees for this lab.

HW12

Closer_num

- I started with the cases stated in the assignment: Num1 is closer and Num2 is closer.
- I realized that when the two numbers are the same distance apart, the function would return nothing so I added a case to handle for equality

- I was satisfied with my work until I accidentally typed a letter into the function then ran it, which returned an error
- Therefore I decided to use a try / except block to handle for any errors that occur because of wrong data types being inputted

Grade_conv

- I started with the cases stated in the assignment: 90-100, 80-89, 70-79, 65-69, and 0-64
- Since the question asked for A to be 90-100 and F to be 0-64 I decided to handle for cases above 100 and below 0
- During class I realized that the function that I had made already didn't handle border functions already so I changed it
- Learning from my experience with the previous function I added a try / except block to handle errors caused by invalid data

Pass_judgement

- I started with the original cases stated in the assignment: A, B, C, D, and F
- I was wondering why using something like "a" would return None, and then I realized that python differentiates between uppercase and lowercase letters.
- I used .upper() to make the input uppercase since I didn't want to add 5 more cases.
- At the end I used else so I could let the user know if there was a wrong or unhandled input.

Fact_r

- At first I forgot how to do recursion. 🤦
- I wrote the function that would return the input multiplied by the function for the number below.
- I added something to check if the value was equal to 1 so it would just return the value.
- I then realized that decimal factorials could cause an infinite loop and therefore a MemoryError so I decided to change it to check if the input was less than or equal to 1.

- I also rounded the input in the function.
- And then from previous experience I added a try / except block so it wouldn't break my testing.

Fact_w

- I defined the output variable beforehand since I didn't want to deal with local vs global variables.
- I set i to input so it would iterate i times.
- I also rounded the input in the function.
- And then from previous experience I added a try / except block so it wouldn't break my testing.

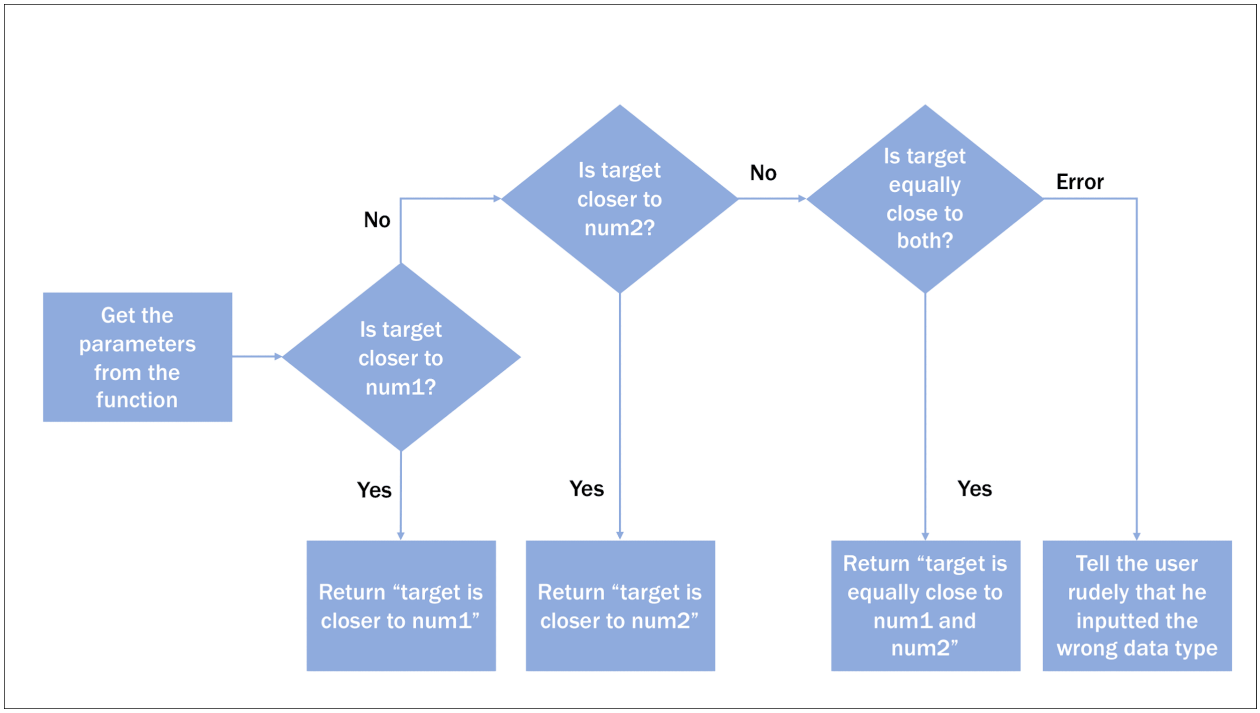
Post-HW12 Endeavors

- I could try implementing a for based factorial function.
- I could also try making a towers of Hanoi function in Python using recursion or while loops. I would like to do this because I was able to do this in Racket, which in many ways is a more basic language than python.

Algorithms and Code

(Algorithms in plain English are WEIRD)

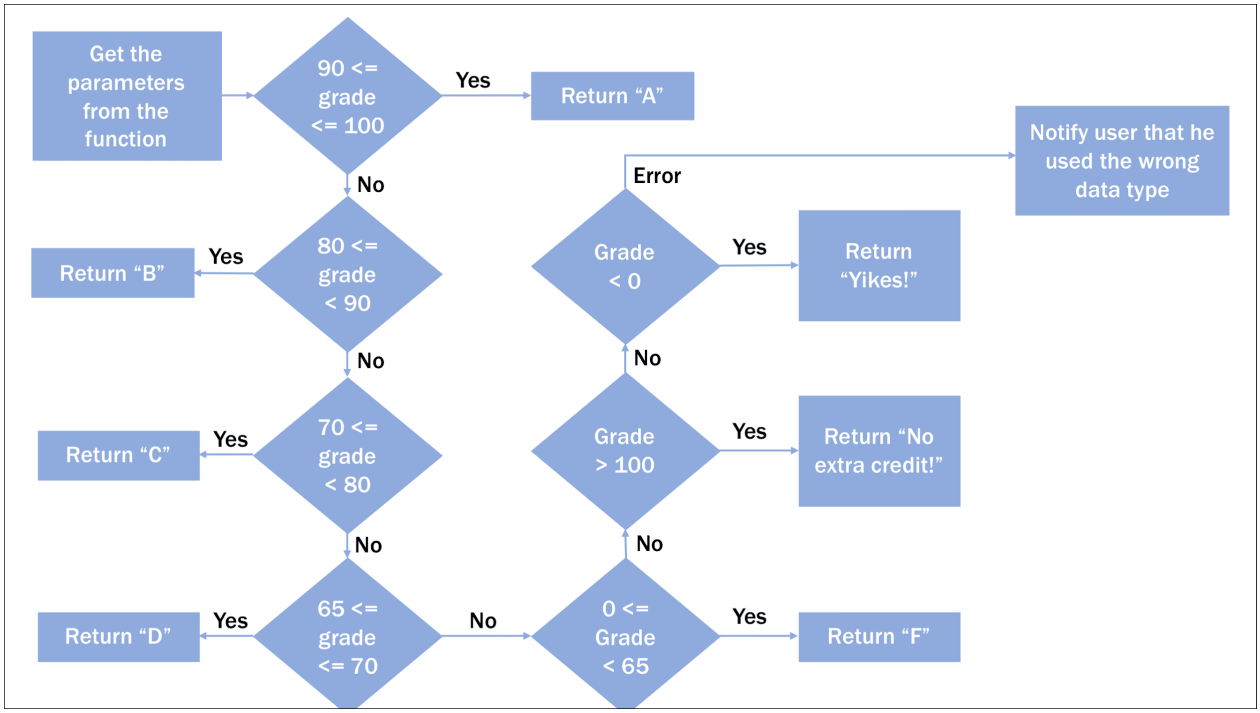
Closer_num




```
def closer_num(target, num1, num2):
    try:
        diff1 = abs(target - num1)
        diff2 = abs(target - num2)
        if diff1 < diff2:
            return str(target) + " is closer to " + str(num1)
        elif diff2 < diff1:
            return str(target) + " is closer to " + str(num2)
        else:
            return str(target) + " is the same distance between " + str(num1) + "
and " + str(num2)
    except:
        return ("Invalid Data Type")
```

To find the closer number to the target, I get the parameters from the function. After, I check if it is closer to num1 by comparing the differences of the distances. If it is, I return a string stating that the target is closer to num1. If it isn’t, I check if it is closer to num2. If it is, I return a string stating that the target is closer to num2. If it isn’t closer to either, I check if it is equally close to both numbers. If it is, I return a string stating that target is equally close to num1 and num2. If there is an error, I let the user know that there was an error, probably caused by inputting an invalid data type.

Grade_conv

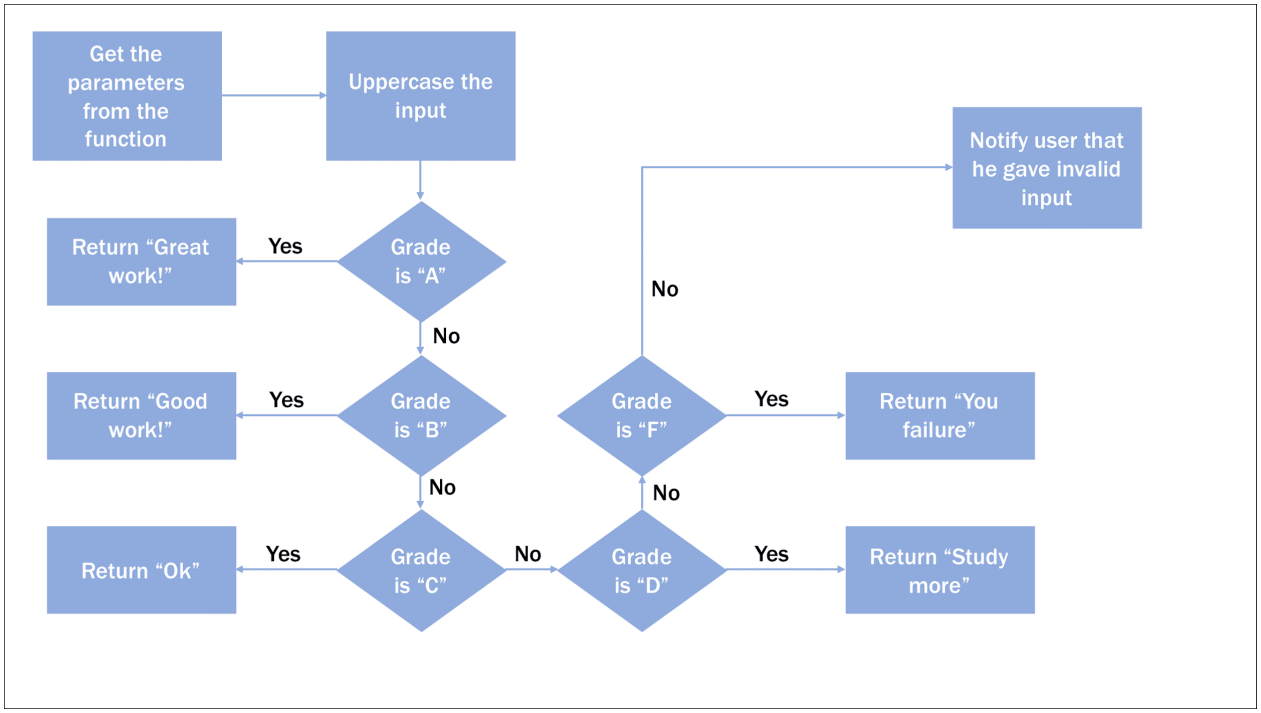



```
def grade_conv(g):
    try:
        if g >= 90 and g <= 100:
            return "A"
        elif g >= 80 and g < 90:
            return "B"
        elif g >= 70 and g < 80:
            return "C"
        elif g >= 65 and g < 70:
            return "D"
        elif g >= 0 and g < 65:
            return "F"
        elif g < 0:
            return "Yikes! That's a bad grade!"
        elif g > 100:
            return "This class doesn't have extra credit!"
    except:
        return ("Invalid Data Type")
```

To convert a numerical grade to a letter, I first get the parameter from the function. Then, I check whether it is between 90 and 100, inclusive. If it is I return A. If it isn’t, I

check whether the input is within 80 and 90. If it is, I return B. If it isn't, I check if the input is between 70 and 80. If it is, I return C. If it isn't, I check if the input is between 65 and 70. If it is, I return D. If it isn't, I check if the input is between 0 and 65. If it is, I return F. If the value if greater than 100, I let the user know that there isn't any extra credit. If the value is less than 0, I let the user know that he shouldn't know that. And if there is an error, I let the user know that he gave an invalid parameter.

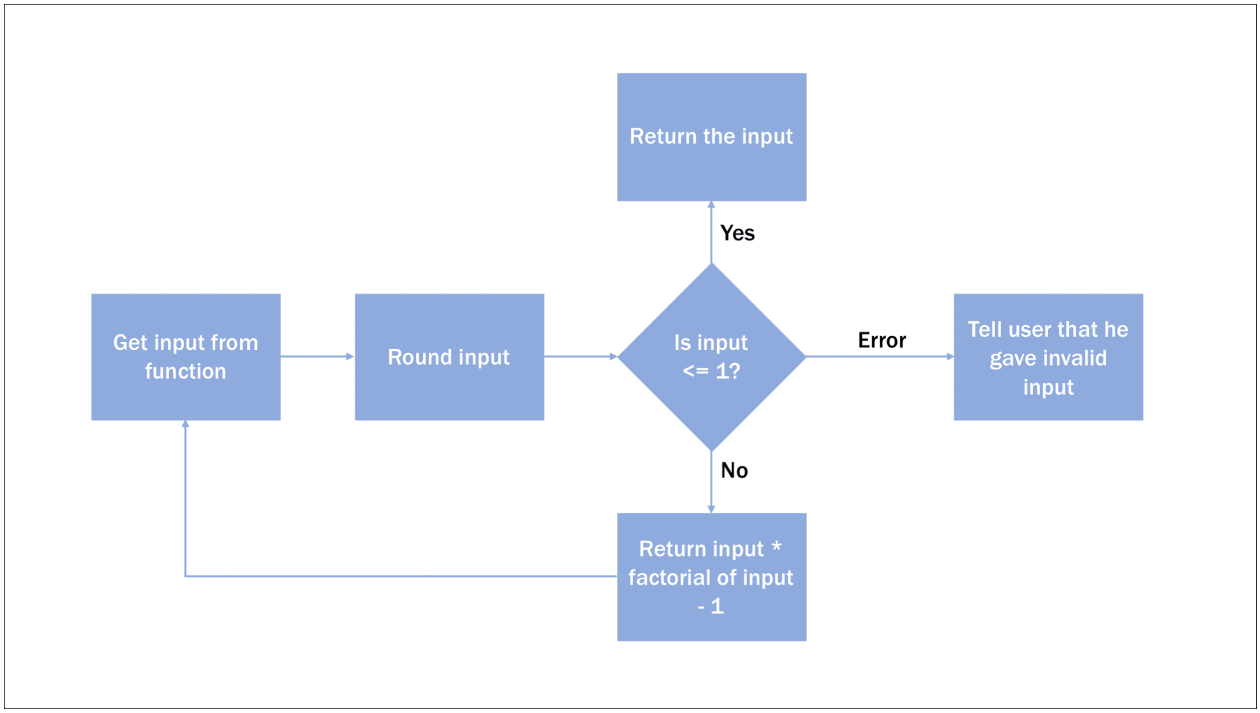
Pass_judgement




```
def pass_judgement(letter_grade):
    grade = str(letter_grade).upper()
    if grade == "A":
        return "Great job! You got an A!"
    elif grade == "B":
        return "Good job. You got a B"
    elif grade == "C":
        return "That's ok, just study more next time. You got a C"
    elif grade == "D":
        return "Try harder next time, you got a D"
    elif grade == "F":
        return "You absolute failure, you got an F!"
    else:
        return ("Invalid Input")
```

I give feedback based on letter grades by first getting the letter grade from the function. Then, I capitalize the input so I can handle lowercase letters or uppercase letters. After, I check if the letter given was A. If it was, I tell the user that he did great work. If it isn't, I check if the letter given was B. If it was, I tell the user that he did good work. If it isn't, I check if the letter given was C. If it was, I tell the user that he did ok. If it isn't, I check if the letter given is D. If it is, I tell the user that he needs to study harder. If it isn't, I check if the letter given was an F. It it is, I tell the user that he did miserably and that he is a failure. If it is none of these letters, I tell the user that he gave invalid input.

Fact_r

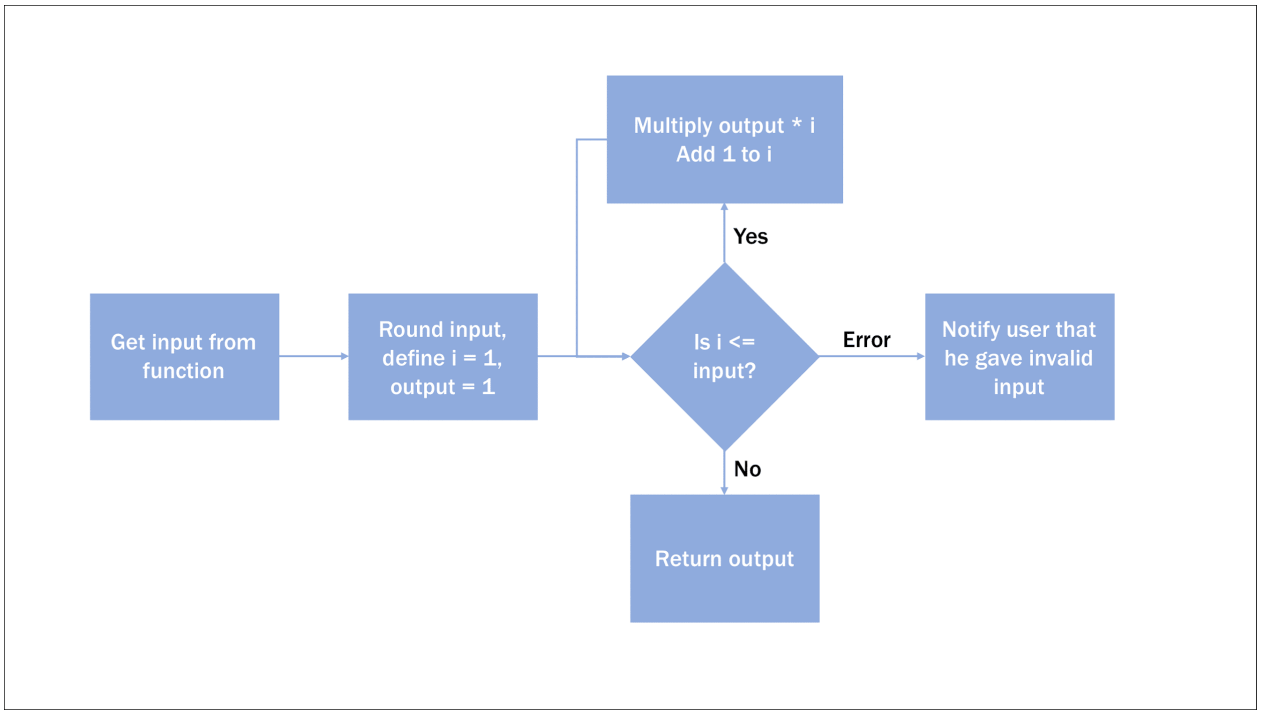



```
def fact_r(input1):
    try:
        input = round(input1)
        if input <= 1:
            return input
        else:
            return input * fact_r(input-1)
    except:
```

```
        print("invalid input")
```

I find the factorial of the input by first getting the parameter from the data. Then, I round the input. Then, I check if the input is less than or equal to 1. If it is, I simply return the input. If it isn't, I return the input multiplied by the factorial of one less than the input. If there is an error, I let the user know that there is an error and tell him that it is probably because he gave invalid input.

Fact_w




```
def fact_w(input1):
    try:
        i = 1
        input = round(input1)
        output = 1
        while i <= input:
            output *= i
            i += 1
        return output
    except:
        print("invalid input")
```

I find the factorial of the input by getting the input from the function. Then I round the input. Then, I define i and output to be 1. After, I multiply output by i and add 1 to i until I is greater than the rounded input. Then I return output. If there is an error I let the user know that he gave invalid data.

Discoveries

- For loops do not work the same way as they do in other languages, instead, they iterate over every unit in an input (for example, each character in “python”)
- When making a decision tree, when representing a conditional statement, you display it as a diamond.
- In python, functions are recorded and stored before any functions are actually called. I know this because you can call the function from inside the function.
- Try / Except blocks help you catch errors when you execute code. For example, as used here, I used them to catch TypeErrors.
- Actually, while blocks can be used to do other things than just iterate a certain number of times. Instead they can be used to do something while something is true (for example do something while a key is down)
- You can concatenate some data types to other data types by using the abbreviation of that data type as a function on the target. However I avoided this because I didn’t think it was a good idea to add extra cases and clutter up the code.
- It’s better to use a while loop instead of recursion since doing recursion, at least in our case, forced the computer to allocate much more memory and took more time (on repl.it, fact_r(1000) gives a memory error [displays as invalid input since we used a catch-all try/except block] but fact_w(1000) computes fine)
- Python, when asked to print an empty string, prints “None” instead of “”.

Unresolved Mysteries

- Why is python indent-based instead of semicolon / bracket based? It’s harder to modify the code in case of mistakes (or if you copy from trash overflow).

- Is there a way to catch specific errors with python? (I can look this one up obviously but we need something to fill this area