Notes - Wednesday 1/6/21

1/6/21 Wed

Other Useful Operators and Functions

Range Operator

Often, we'll have lists of numbers and instead of typing out all the numbers in the list, we can use the Range function. This works similar to the slicing function mentioned above. So let's say I want to print out all the numbers in a range. I could setup a list:

`num_list = [0,1,2,3,4,5,6,7,8,9]`

Or I could use range. Remember that like in a slice, the range ends one value before the one you define. So if I put 10 into the range, Python counts up to 9. The basic syntax is:

` range (number-1)`

You can also specify where you're starting the range. Syntax is:

`range (start, end-1)`

You can also add in a step. Syntax is:

`range (start,end-1,step)`

Or you can get a list of numbers with a range using:

`list(range(start,end-1,step))`

Enumerate Function

Takes any iterable object and returns an index counter and the object or elements. So let's say you want to keep track of how many times a Loop runs. This is better explained with examples:

This is the example before I learned `Enumerate`:

```
# enumerate function
# I want to count the loops happening
# I could setup this counting thing
# but we do this enough in Python that we have enumerate

index_count = 0

for letter in 'abcde':
    print('At index {} the letter is {}'.format(index_count,letter))
    index_count += 1
```

Now let's simplify the example with `Enumerate`

```
# This counts the loops using enumerate and outputs tuples

index_count = 0
word = 'abcde'

for letter in enumerate(word):
  print(letter)
```

Zip Function

This puts lists together. The basic syntax is `zip`. So the objects in the list will be determined by the list with the least amount of functions. The output is a tuple but the tuple only shows if you run something. Otherwise it acts like a normal list. You can also do tuple unpacking.

Again this makes more sense looking at examples:

```
zip_list1 = [1,2,3,4,5,6]
zip_list2 = ['a','b','c']
zip_list3 = [100,200,300]

for item in zip(zip_list1,zip_list2,zip_list3):
    print(item)
```

This is the above output but in a list form:

`list(zip(zip_list1,zip_list2,zip_list3))`

And here's zip with tuple unpacking:

```
for a,b,c in zip(zip_list1,zip_list2,zip_list3):
    print(b)
```

In Operator

We can use this to check if something is in a list, strings, dictionaries. It will output a boolean True or False. Basic syntax is:

`thing_2_check in [ur_object_here]`

List: Is 2 in a list of 1,2,3? This will output True. If I swap 2 for 'x', it will output False.

`2 in [1,2,3]`

String: Is 'a' in 'a world'? This will output True.

` 'a' in 'a world'`

Dictionary: I can check for keys or values. I setup a dictionary then need to add `.values()` or `.keys()`

```
d = {'k1':123}

123 in d.values()
```

Min/Max

There's a built in Min and Max function to check the min and max in a list. Syntax is `(min(list))`` and` (max(list))```

Random Library

There's a library called random which has functions. These functions can generate random numbers. The basic syntax is:

`from random import function`

Select Random Functions:

1. Shuffle - Scrambles a list. So for example let's scramble a list and then output the new scrambled list:

```
mylist = [1,2,3,4]
from random import shuffle
shuffle(mylist)
mylist
```

2. Random number generator - Randint will generate a random number within a range. The basic syntax is below:

```
random import randint
randint(x,y)
```

Input Function

This let's you accept user input. The basic syntax is `input(ur_input_here)`. To save the input, you can cast it as a variable. So the example syntax would be `result = input(ur_input_here)`.

Result would be a string though. So you'd want to then cast the result as another type. So the example would be:

`result = int(input('Fav number:'))`

Temperature Converter Program

After I got to the end of the Udemy course and it was testing time, I tried to write a program that converts temperature into Celsius and Fahrenheit. I used this code as the starting point.

Pieces of code I'll need to understand the notes going forward:

1. Temperature Converter Baby Code.py - For base baby code
2. Temp Convert Phil 1.py - For Step 1 code
3. Temp Convert Phil 2.py - For Step 2 code
4. Temp Convert Phil 2 - Anson Mod Final.py - For Step 2 code - Final Code

My base baby code

So here's the basic code I made using the starting point. I added in the temperature rounding and the Unit checking.

```
# Intro Words
# This is a beginner program for Anson to learn how to make a simple
# temperature conversion program. This will convert from C to F and in reverse

temp = float(input("Enter temperature = "))
unit = input("Enter C or F (for Celsius or Farhenheit) = ")



if unit == "F" or unit == "f":
    fahrenheit = (temp * 9/5) + 32
    print(f'{round(fahrenheit,2)} F')
elif unit == "C" or unit == "c":
    celsius = (temp - 32) * 5/9
    print(f'{round(celsius,2)} C')
elif unit != "F" or unit != "f" or unit != "C" or unit != "c":
    print("Not a valid unit, try again.")
```

Next steps

1. I want to have the code check for errors. So if I put in temp = asf and unit = F, I want it to tell me I'm bad.
2. I want the code to loop through 3 times. So if I make mistakes, I want it to re-prompt me again to input my values. After 3 guesses, it quits out.

Step 1 Code

The base code is "Temperature Converter Baby Code.py". Then I wanted to add in mistake proofing so if I put in `asdf` as the temperature, it would say "Oh you messed up, try again" and reprompt me. Jwaz nerd chat introduced me to "try/except" which is explained nicely here. And they also introduced me to `NameError` and `ValueError` which is explained here.

Try/Except doesn't won't set anything if it doesn't work though. It only sets the variable if it works. So I'm getting a NameError with this code I came up with for Step 1 (I got the idea for checking if the value is a float from here):

```
# Intro Words
# This is a beginner program for Anson to learn how to make a simple
# temperature conversion program. This will convert from C to F and in reverse


temp = input("Enter temperature = ")
unit = input("Enter C or F (for Celsius or Farhenheit) = ")


try:
    temp_float = float(temp)
except ValueError:
    print("Not a valid temp")
 while temp_float == float:
    if unit == "F" or unit == "f":
        fahrenheit = (temp_float * 9/5) + 32
        print(f'{round(fahrenheit,2)} F')
    elif unit == "C" or unit == "c":
        celsius = (temp_float - 32) * 5/9
        print(f'{round(celsius,2)} C')
    elif unit != "F" or unit != "f" or unit != "C" or unit != "c":
        print("Not a valid unit. Try again.")
if temp_float != float:
    print("You messed up. Try again.")
```

Phil helped me with the code "Temp Convert Phil.py". His code looks like:

```
temp = (input("Enter temperature = "))
unit = input("Enter C or F (for Celsius or Farhenheit) = ")


try:
    temp_float = float(temp)
except ValueError:
    print("Not a valid temp")
    quit()

if not temp_float:
    print("You messed up. Try again.")
    quit()

if unit.lower() == "f":
    fahrenheit = (temp_float * 9/5) + 32
    print(f'{round(fahrenheit,2)} F')
elif unit.lower() == "c":
    celsius = (temp_float - 32) * 5/9
    print(f'{round(celsius,2)} C')
else:
    print("You messed up. Try again.")
```

Phil's notes on the code:

1. He taught me that `quit()` or `exit()` are things that exist. So now the code just quits out when I put in the wrong values.
2. A while loop I would reach for if I had more than one thing to iterate through, but in this situation we have 1 unique thing to make a decision on 1 time. I might use a while if I had 1000 temperatures but I couldn't be sure how many times I needed to go through the processing for conversion
3. The if elif else is the right way to think abou tthe linear decision process, if one thing, otherwise if another thing, otherwise here's what to do when you're out of options
4. Finally, because in my solution I didn't use a loop of any kind, pass doesn't work for this either. In a loop a pass would be proper if you needed a reason to exit the loop but continue to the logic after the loop. Instead, when you need to exit the whole program, exit() or quit() would be better tools

Step 2 Code

Now I want the code to repeat itself. I want the code to allow me 3 tries before quitting out. I had the ideal of using true or false booleans and while loops but wasn't sure how to bridge the gap. Phil helped with this code called "Temp Convert Phil 2.py":

```
# Instantiate initial true flag to enter loop
run_loop = True
```

```
# While set to go
while run_loop:
    # Try to capture a float at input time so we don't have to parse it later
    try:
        temp = (float(input("Enter temperature = ")))
    except Exception:
        print("Input isn't a temperature; try again")
        # If we can't establish a float for the first input, we'll simply skip the rest of this iteration and never set
        # the run_loop flag to false, allowing loop to continue
        # Exception catches all errors
        # more info here: https://docs.python.org/3/library/exceptions.html
        continue

    # Instantiate a sub loop
    run_loop_sub = True

    while run_loop_sub:
        try:
            unit = str(input("Enter C or F (for Celsius or Farhenheit) = "))
        except Exception:
            print("Input needs to be c/C or f/F")
            # Harder to hit this since "" is a string in input, but if it fails for whatever reason
            # just try again
            continue

        if unit.lower() == "f":
            fahrenheit = (temp * 9/5) + 32
            print(f'{round(fahrenheit,2)} C')
            # Completion condition met, set loop flag to false to exit loop after this iteration
            run_loop_sub = False

        elif unit.lower() == "c":
            celsius = (temp - 32) * 5/9
            print(f'{round(celsius,2)} F')
            # Completion condition met, set loop flag to false to exit loop after this iteration
            run_loop_sub= False

        else:
            print("You need to enter either c/C or f/F")
            # There is no satisfactory completion here, so don't set the close flag


        # If we make it here, that means that the sub while loop was satisfied, and there is no further exceptions to
        # skip this flag; we can probably end the loop
        run_loop = False
```

Notes on the code:

1. Instantiate to represent an instance
2. Try/Except has the `Exception` class which catches all built-in, non-system existing exceptions. This is what I wanted when I was messing with `ValueError` and `NameError` above.
3. Instead of the `!=` I was using to check for the case, the way Phil handled it was to just parse the input into a string then uses `unit.lower()` to set the case to lower case only. That way it doesn't matter what the case is.
4. Comments are good
5. Loops and subloops are a thing. Could have a overall loop and loops underneath the overall loop

So now I got most of the code working. "Temp Convert Phil 2.py" aka the code above will keep looping until it gets the correct answer. But now I want to add in the 3 strikes or the code ends. So my code is found in "Temp Convert Phil 2 - Anson Mod Final.py". It looks like:

```
# Instantiate initial true flag to enter loop
run_loop = True
# Set global variable to count
retry = 0
# Runs code while retry is under 3
while retry < 3:


    # While set to go
    while run_loop:
        # Try to capture a float at input time so we don't have to parse it later
        try:
            temp = (float(input("Enter temperature = ")))
        except Exception:
            # Exception catches all errors
            # more info here: https://docs.python.org/3/library/exceptions.html
            print("Input isn't a temperature; try again. Max 3 attempts. Attempts:",retry+1)
            retry += 1
            # If we can't establish a float for the first input, we'll simply skip the rest of this iteration and never set
            # the run_loop flag to false, allowing loop to continue
            # Print is setup so it tells me how many attempts I'm at and shows the count
            break
            # Ends the program if I guess too much

        # Instantiate a sub loop
        run_loop_sub = True

        while run_loop_sub:
            try:
                unit = str(input("Enter C or F (for Celsius or Farhenheit) = "))
            except Exception:
                print("Input needs to be c/C or f/F")
                # Harder to hit this since "" is a string in input, but if it fails for whatever reason
                # just try again
                continue

            if unit.lower() == "c":
                fahrenheit = (temp * 9/5) + 32
                print(f'{round(fahrenheit,2)} F')
                print('You know the temp now!')
                # Completion condition met, set loop flag to false to exit loop after this iteration
                run_loop_sub = False

            elif unit.lower() == "f":
                celsius = (temp - 32) * 5/9
                print(f'{round(celsius,2)} C')
                print('You know the temp now!')
                # Completion condition met, set loop flag to false to exit loop after this iteration
                run_loop_sub= False

            else:
                print("You need to enter either c/C or f/F")
                # There is no satisfactory completion here, so don't set the close flag

            # If we make it here, that means that the sub while loop was satisfied, and there is no further exceptions
to
            # skip this flag; we can probably end the loop
            run_loop = False
```

```
#if retry == 3:
#    run_loop = False
    # Not sure if this helps or hurts
    # After experimenting, it doesn't seem to matter if it's here
```

Notes on the code:

1. There's a hang up, the code doesnt close cleanly like it did in "Temp Convert Phil 2.py" after the temperature is input correct. But it does work.
2. When the 3 guesses are put in, so "asdf" for the temperature, I created a print that warns how many attmepts are left and what attempt number we're on.
3. The last 4 lines of code don't seem to do much. Not sure why I can just run a while loop as is.
4. Learned what a global variable is.
5. Learned what a counter is and how break works.

Revision #5
Created 7 January 2021 04:10:34 by cba88
Updated 7 January 2021 05:04:07 by cba88