Laboratory 3

September 10 and 12, 2019

Conditional Statements and For Loops

1. A program that asks the user for input, the strength of an earthquake, and then prints out a string according to the following table. Values below 1 are not on the Richter scale and should result in an error message.

Magnitude	Description
[1, 2)	Micro
[2, 4)	Minor
[4, 5)	Light
[5, 6)	Moderate
[6, 7)	Strong
[7, 8)	Major
[8, ∞)	Great

2. Heron's method calculates an approximation to the square-root. If S is the number whose square-root is to be taken, and a is an approximation of the square-root, then Heron's method calculates a better approximation by

$$a_{\text{better}} = \frac{1}{2}\left(a + \frac{S}{a}\right)$$

For example, if $S = 2$ and we choose $a = 1$, then the better approximation according to Heron's is 1.5. If we apply the method again, then we get 1.4166666666666665. Another application gives us 1.4142156862745097, which is already accurate to the fifth digit after

the decimal. Write a program that asks the user for a number. Calculate an approximation of the square-root by starting out with initial approximation 1 and then running Heron's method twenty times to get a better approximation. (Hint: You need to use a for loop. In the body of the for-loop, you improve the value of a.)

```
a = 1
for iter in range(20):
     a = better_value_for_a
print(a)
```

Notice that we do not use the iteration variable `iter`.

3. We can use for-loops in order to calculate finite sums and products. For instance, in order to calculate the sum

$$\sum_{\nu=1}^{n} (\nu^4) = 1^4 + 2^4 + 3^4 + \ldots + (n-1)^4 + n^4$$

we can use a programming pattern centered around an accumulator that contains partial sums. Initially, the accumulator is zero. We then generate the numbers $\nu = 1,2,3,\ldots$ in turn and add the addend ν^4 to the accumulator. The resulting accumulator at the end of the for-loop is the sum. The code fragment below gives the code. The lines are numbered on the right. In line 1, we just asks the user to enter the range of the summation. Notice

```
n = int(input("Enter n: "))                              #1
accumulator=0                                            #2
for i in range(1,n+1):                                   #3
    accumulator += i**4                                  #4
    #print("i", i, "accumulator:", accumulator)          #5
print("The result is", accumulator)                      #6
```

again the need to convert the string returned by the input-function into an integer. The second line initializes the accumulator. Line 3 gives the for loop. Notice the bounds. The first value for *i* has to be 1, the last one *n*, meaning that the stop value is *n*+1. Many people are disturbed by Python using the stop-value (the first value **not** taken), but once you understand a bit more of the language, it actually does make sense. Line 4 then updates the accumulator. The += is just a short-cut notation for

`accumulator = accumulator + i**4.`

Line 5 is commented out and is used for debugging. It prints out the progress in the for-loop. Finally, Line 6 presents the result. Notice that it is dedented as it is not part of the for-loop.

Tasks:

A. Calculate $\sum_{i=0}^{1000} i^2$. (The answer is 333 833 500).

B. Calculate $\sum_{i=0}^{1000} \frac{1}{1+i^2}$. (The answer is 2.075 674 547 634 748.)

4. For loops can similarly be used for the calculation of products. However, the accumulator then needs to be initialized to 1 (the neutral element of multiplication). If it were initialized to 0, then updating it by multiplying it with an factor still gives you 0.

   Tasks:

   A. Calculate $\prod_{i=1}^{1000} \frac{2i+1}{2i}$. (The answer is approximately 35.696).

   B. Calculate $\prod_{i=1}^{100} \frac{i^2+1}{i+2}$. (The answer is approximately 6.5944×10^{154}).

Approximations for π

5. There are many formulas that allow us to calculate the mathematical constant π. Test out the following product and sums for π by evaluating the first 100, 1000, and 10000 members of the sums or products.

   Eulers Formula: $\frac{\pi^2}{6} = \sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \ldots$

   Plouffe's BBP digit extraction algorithm:

$$\pi = \sum_{k=0}^{\infty} \frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right)$$

   Plouffe's algorithm is very cool because you can calculate hexadecimal digits for π without reference to previous ones. This might be your only class you ever take at Marquette where you are the beneficiary of a Mathematical result presented while you were almost born!

   You can find the value for π as `math.pi`.