Unstructured challenges

# Essentials

**Objective:** tests one’s proficiency with the essential components of programming, and Python’s syntax of them:

* Variables
* Conditional statements
* Iteration (in particular, the indefinite WHILE loop)

## 1 Guess-my-number game

**Summary:** Player keeps guessing a number until the guess is correct

Write a Python script that does the following: A computer “thinks” of a whole number between 1 and 100, and invites the user to guess it. Every time the user guesses a number, the computer tells if it’s too high, too low, or just right. The game ends once the user correctly guesses the number.

FYI

To generate a random number between 1 and 100, include the following command at the beginning of the script:

**from random import randint**

For the rest of the script, calling the function

**randint(1,100)**

will result in a whole number between 1 and 100.

## 2 Practice 3-digit multiplication

**Summary:** app to practice the guess-and-check method for multiplying large numbers

Write a program that gives the user two 3-digit numbers and asks the user to multiply them. After each guess, the program gives feedback: too high, too low, just right. The program runs until the user guesses correctly.

FYI: this is very much like the Guess-my-number diagnostic.

## 3 Square root

**Summary:** Find the square root of a two-digit number

The program gives a 2-digit number and asks the user to guess its square root accurately to within two decimal digits. After each guess, the program gives feedback: too high, too low, just right. The program runs until the user guesses a number that is close enough to the actual square root.

FYI: like Guess-my-number, but with real numbers, so requires knowledge of **float** type variables.

## 4 [Collatz Conjecture](https://en.wikipedia.org/wiki/Collatz_conjecture)

**Summary:** How long before this sequence ends in 1?

Here are the rules for the Collatz sequence: starting with a whole positive number bigger than 1,

* if the number is odd, multiply by 3 and add 1
* If the number is even, divide by 2

The sequence ends when the number reaches 1

Ask the user for a whole positive number. Starting with that number, print out the Syracuse sequence. Then report how many steps it took to reach 1.

More in-depth video: [The Simplest Math Problem No One Can Solve - Collatz Conjecture](https://www.youtube.com/watch?v=094y1Z2wpJg)Watch NOT on work time, but it is fascinating!

# 

## 5 Very simple calculator

**Summary:** Adds, subtracts, multiplies, divides

Write a very basic calculator that can add, subtract, multiply, or divide two numbers. This program mimics in text a cheap grade-school calculator. Ask for the first number. Then, ask the user which operation to perform (+, -, \*, /, or q to quit). Then ask for the second number. Print the result, and ask which operation the user wants to perform next (and then get the next number). Keep going until the user specifies “q” to quit.

# Lists

**Objective:** tests one’s proficiency with Python lists

## 1 Basic stats

**Summary:** Gets numbers from the user. Returns their minimum, maximum, and median.

**Basic version:**

Ask the user to input a bunch of numbers (hint: unless you are good at strings, have the input be a while loop that will not quit unless the user types a (specific) letter). Determine and print the smallest number (minimum) and the largest number (maximum).

Also have it print the median number. For the median number: if you rearrange the numbers from lowest to highest, that’s the number in the middle (or, if two numbers are in the middle, then it’s the average of those two).

(Hint: look up how to sort a list.)

**More advanced**:

Find the user’s mean (the average of all the numbers) and the mode (the most common number(s))

**Hint:** To help you with this (and math class), try remembering the following mnemonic: Median in the middle, the mean old average, and the *a la mode*! It has helped me (Adrian), but it may not work for everyone.

## 2 Pronounce the number

**Summary:** Gets an integer from the user. Prints out how to say this integer in English.

**Example:**

User inputs 12345, the program outputs “twelve thousand three hundred and forty five”.

**Basic version:**

Ask the user to input a two-digit whole number.

**More advanced versions:**

1. Three-digit whole numbers
2. Six-digit whole numbers

**Hint:** you could start with a list of the names of single-digit integers, like:

singles = [“zero”, “one”, “two”, “three”, “four”, “five”, “six”, “seven”, “eight”, “nine”]

Then singles[3] is the name for 3. But how will you extend it to two-digit numbers?

# 

# Dictionaries

**Objective:** Tests one’s proficiency with Python dictionaries

## 1 Scoreboard

**Summary:** Create a fictional scoreboard that will allow users to input their own names and scores to the scoreboard.

**Basic version:**

The program should ask the user to type their name and their score, saving that name and score to the proper spot (if it is the highest score, it should go on top, second highest below the first on, etc.) then print the completed dictionary.

**Advanced ideas:**

Remove scores so there are a total of 5 high scores in the dictionary.

If the user’s name already appears in the dictionary, ask if they are the same person. If so, keep their highest score. If not, ask them to use a different username.