AoPS Online
Art of Problem Solving

## Syllabus — Introduction to Programming with Python

Introduction to Programming with Python is a first course in computer programming using the Python programming language. This course covers basic programming concepts such as variables, data types, iteration, flow of control, input/output, and functions.

This course is specifically designed for high-performing students and draws material from many programs for top middle and high school students in the country. Our philosophy is that students develop more by learning to solve problems they haven't seen before, as opposed to offering repeated drills that students can memorize their way through. In this way, our classes are structured much more like courses at top-tier colleges.

**Book:** This class uses Chapters 1-10 in a version of *How to Think Like a Computer Scientist: Learning with Python 3*, by Wentworth, Elkner, Downey, and Meyers.

**Time Commitment:** This 12-week course includes 1.5 in-class hours and at least 3-5 hours/week programming outside of class, corresponding to a 1/2-year course.

**Grading:** $32\%$ Short-Answer Challenge Problems, $64\%$ Python Coding Problems, and $4\%$ Class Participation.

**Content:**

| Week | Topic |
|---|---|
| 1 | What is Programming? What is Python? |
| 2 | Data Types, Variables, and Expressions |
| 3 | Turtle Graphics |
| 4 | Functions |
| 5 | Conditionals |
| 6 | Flow of Control |
| 7 | Strings |
| 8 | Lists and Tuples |
| 9 | File Input/Output |
| 10 | Dictionaries |
| 11 | Final Project, Week 1 |
| 12 | Final Project, Week 2 |

**Sample Problems:**

- Write a program that simulates rolling two 6-sided dice 1000 times, and keeps track of how many times each sum occurs. Then your program should print out a table of the number of times it rolled each sum from 2 through 12.

- Write the function `sum_of_proper_divs()` as described below and then use it to solve the following problem:

  Function `sum_of_proper_divs(n)` returns the sum of all of the proper divisors of $n$. For example, `sum_of_proper_divs(12)` returns 16, because $1 + 2 + 3 + 4 + 6 = 16$.

  A number $n$ is called *perfect* if the sum of its proper divisors equals $n$. For example, 6 is perfect because its proper divisors are 1, 2, and 3, and $1 + 2 + 3 = 6$, and 28 is perfect because its proper divisors are 1, 2, 4, 7, and 14, and $1 + 2 + 4 + 7 + 14 = 28$.

  6 and 28 are the only perfect numbers less than 100. There is only one 3-digit perfect number. Find it.

  Since we are using `sum_of_proper_divs()` to solve this problem, please include test cases or otherwise describe how you verified that it is working correctly.

- Mastermind is a game for 2 players. The commercial version of the game uses six colors, which vary from game to game; for our purposes we'll use red, orange, yellow, green, blue, and purple. One player (the **codemaker**) thinks of a secret code, which is any 4 colors in some order (in the easy version of the game, the 4 colors all have to be different; in the harder version, a color is allowed to repeat one or more times). The other player (the **codebreaker**) has 10 attempts to guess the code.

  For each guess, the codemaker tells the codebreaker how many colors are correct and in the correct location, and how many colors are correct but in the wrong location. The guesser is not told exactly which colors are correct. In the commercial version of the game, black and white pegs are used for this: a black peg means a color correct and in the correct location, a white peg means a color correct but not in the correct location. You can use black and white as a shorthand for your responses.

The most common mistake students make in this project is miscalculating the number of black and white pegs, so be sure to understand the instructions and the following examples. It is important for students to thoroughly test this part of the program. For the following examples, suppose the code is red-blue-red-yellow.

If a guess is green-blue-purple-purple, the response would be **1 black 0 white** (meaning that one peg—the blue one—is correct and in the correct location, and the rest are incorrect).

If a guess is red-yellow-green-purple, the response would be **1 black 1 white** (the red is correct and in the correct location, the yellow is correct but in the wrong location).

If a guess is red-red-blue-orange, the response would be **1 black 2 white**.

If a guess is blue-green-blue-red, the response would be **0 black 2 white** (only one of the two blue guessed gets a white peg, since there is only one blue in the actual code).

Note that black takes precedence over white, so if a guess if green-blue-blue-green, the response

 

would be **1 black 0 white**. (The blue guess that matches location takes precedence over the blue guess that doesn't.)

Implement this game where the computer randomly generates a code, and the user plays as the codebreaker. The codebreaker wins if he or she gets the code in 10 guesses or less. You can decide whether to implement the easy or hard version of the game (or both).

A sample run:

```
I've got a code of length 4
It uses colors in roygbp
Guess #1 -- enter your guess: rybg
1 black 1 white
Guess #2 -- enter your guess: ryop
2 black 1 white
Guess #3 -- enter your guess: rpyo
1 black 2 white
Guess #4 -- enter your guess: ryog
1 black 2 white
Guess #5 -- enter your guess: ropy
2 black 1 white
Guess #6 -- enter your guess: ropg
2 black 2 white
Guess #7 -- enter your guess: rogp
You cracked the code!
```