AoPS Online
Art of Problem Solving

Syllabus — Intermediate Programming with Python

The first half of Intermediate Programming with Python covers recursion and object-oriented programming concepts, including classes, methods, and inheritance. In the second half of the course, students apply these concepts to programming graphical user interfaces with event-driven programming.

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 11-16 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	Review of Python Basics and Programming Basics
2	Recursion
3	Classes and Object-Oriented Programming (OOP)
4	More OOP
5	Still More OOP!
6	Inheritance
7	Event-Driven Programming and GUIs
8	More with GUIs
9	Project Week 1
10	Project Week 2
11	Game Design Week 1
12	Game Design Week 2

Sample Problems:

- You've just been named the new director of the United States Mint. For some reason, you've been authorized to do whatever you want with our coinage. You want to simplify our coins using the following criteria:
  - There should be only three denominations of coins. You can keep any three existing denominations or come up with new ones.
  - Assume that in any transaction, the number of cents given as change is equally likely to be any of the hundred quantities from 0 to 99.

  Write a Python program to find the 3 different coin values that minimize the average number of coins given as change.

- There is a well-known puzzle involving a 3-liter jar and a 5-liter jar. We can do 3 things with each jar:
  - Fill the jar: the jar is completely filled with water.
  - Empty the jar: the jar is emptied completely and contains no water.
  - Pour into the other jar: water is poured from jar A into jar B until either: (1) jar A runs out of water and is empty; or (2) jar B becomes full, and jar A contains any leftover water. Notice that no water is lost while pouring, and we are not allowed to pour "halfway" – we must pour until either the source jar is empty or the destination jar is full (whichever happens first).

  The goal is to end up with exactly 4 liters of water in the 5-liter jar.

  Write a new Python class called `Jar` to simulate a jar from this puzzle. You should write a constructor to create a new empty jar (where the capacity of the jar is passed as a parameter), a string conversion method to return the status of a jar (for example, "a 3-liter jar with 2 liters of water"), and a method for each of the three operations listed above.

  Then, write a program that creates two jars of size 3-liters and 5-liters, and perform the necessary operations to result in the 5-liter jar containing exactly 4 liters of water.

- Write a Python program that plays the game of Checkers. Create subclasses of tkinter classes to build your GUI. A player wins if their opponent has no legal moves, either because their opponent has run out of pieces or all of their pieces are blocked.

  A piece may be moved forward to an empty diagonally-adjacent square. If a piece is able to jump, it must do so. A piece is able to jump a piece of the other player if it is on a square the current player's piece could move to if the square were empty, and if the square immediately beyond the opponent's piece (continuing in a straight line) is empty. The player moves their piece to the empty square beyond the opponent's piece and removes the opponent's piece from the board. In addition, if the piece that has just jumped may make additional jumps on the same turn, it must continue jumping until it has no more jumps available or until it becomes a new king.

  If a piece reaches the last row of the board, it immediately becomes a king. A piece becoming a king ends the player's turn, even if the king was made via a jump and is still able to jump. Kings can move and jump forwards or backwards.

  For an optional extra challenge, add a computer player to your Checkers game. The human player should be able to designate one of the two colors to be a computer player.