This handout continues an example that was started in the previous handout.

1 Playing multiple games

We would like to enhance our game so that once a single game is finished, we prompt the user to play to see if he/she wants to play again. Below are two possible ways to add the "play again" feature to our program. Both examples play a game that is even simpler than mastermind: it's a "guess a number" game where the computer selects 13 as the secret number and the user has three chances to guess the secret number.

Version 1

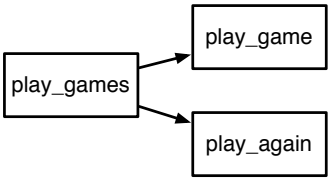
```
def play_again():
    answer = raw_input("Play again? ")
    return answer == 'y'

def play_game():
    secret = '13'
    num_guesses = 0
    while num_guesses < 3:
        guess = raw_input("Guess: ")
        if guess == secret:
            break
        num_guesses += 1
    if guess == secret:
        print "You win!"
    else:
        print "You lose!"

def play_games():
    keep_playing = True
    while keep_playing:
        play_game()
        keep_playing = play_again()
    print "Well, fine, see ya later!"

play_games()
```

Version 2

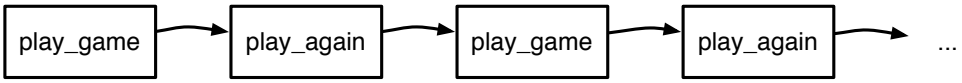
```
def play_again():
    answer = raw_input("Play again? ")
    if answer == 'y':
        play_game()
    print "Well, fine, see ya later!"

def play_game():
    secret = '13'
    num_guesses = 0
    while num_guesses < 3:
        guess = raw_input("Guess: ")
        if guess == secret:
            break
        num_guesses += 1
    if guess == secret:
        print "You win!"
    else:
        print "You lose!"
    play_again()

play_game()
```

Which version has a better program design? The answer is on the back of this handout, but don't peek!

Version 1 is better than Version 2. The main reason is because the functions have a *hierarchical structure*, as shown in this diagram. Each box is a function and there is an arrow from box A to box B if function A calls function B.



In contrast, Version 2 is non-hierarchical:



With Version 2, python never "leaves" the first game. In fact, if you play 4 games and then decide to stop, you will see that `Well, fine, see ya later!` is printed four times!

Furthermore, in Version 2, the `play_game` function not SOFA because it does more than one thing: it not only plays a single game, but it actually initiates the process of playing multiple games.

2 Exercises

Solutions are presented in class and also included in the moodle version of this handout. All of these questions are in the context of the game of mastermind.

1. Write a function `count_exact` that takes in a guess and a secret code and counts the number of exact matches (correct color, correct place) – in other words, red pins.
2. Write a function `count_inexact` that takes in a guess and a secret code and counts the number of inexact matches (correct color, incorrect place) – in other words, white pins. Avoid double counting exact matches!
3. Write a function `is_valid` that takes in a guess, which is a string, and returns `True` if the guess is valid and `False` otherwise. A guess is valid if it contains 4 characters consisting only of the letters R, G, B, Y, P, and O.
4. Write a function `prompt_user` that repeatedly prompts the user for a guess until they enter a valid guess. This function should call `is_valid`. This function should not take any parameters and it should return a string, corresponding to a valid user guess.
5. Write a function `generate_code` that randomly generates a secret code. The secret code should be exactly four characters randomly selected (with replacement) from letters R, G, B, Y, P, and O. Hint: import the `random` module use `random.randint(0,5)` to randomly generate a number between 0 and 5 (inclusive) and use this random number to index into the string `'RGBYPO'`.
6. Put all of these pieces together into a `play_game` function that plays a single game of mastermind. Then write a program that prompts the user to play mastermind and then re-prompts them to play again after the game ends.