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

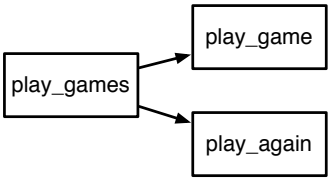
```
def play_again():
    answer = raw_input("Play again? ")
    return answer == 'y'

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

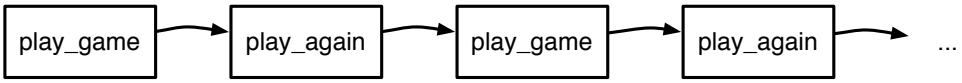
```
def play_again():
    answer = raw_input("Play again? ")
    if answer == 'y':
        play_game()
    print "Well, fine, see ya later!"

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).

Solution:

```
def count_exact(secret, guess):
    '''(str, str) -> int
    Returns number of exact matches between secret
    and guess.  Match is exact if same character
    occurs at the same index.

    Expects guess and secret to have same length.
    >>> count_exact('RBGY', 'RGBY')
    2
    '''
    count = 0
    for i in range(len(secret)):
        if secret[i] == guess[i]:
            count += 1
    return count
```

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). Avoid double counting exact matches!

Solution:

```
def count_inexact(secret, guess):
    '''(str, str) -> int
    Returns number of inexact matches between secret
    and guess.  Match is inexact if same character
    occurs but at a different position.

    Expects guess and secret to have same length.
    >>> count_inexact('RBGY', 'RGBY')
    2
    >>> count_inexact('GOOO', 'GPPG')
    0
    >>> count_inexact('GOOO', 'PGGP')
    1
    >>> count_inexact('RYOP', 'RGGG')
    0
    >>> count_inexact('RRGB', 'YYRY')
    1
    >>> count_inexact('RYOP', 'GGGR')
    1
    >>> count_inexact('RRRG','RGGG')
    0
    '''
    secret = list(secret)
    guess = list(guess)

    # mark exact matches
    for i in range(len(secret)):
        if secret[i] == guess[i]:
            secret[i] = '!'  # mark secret
            guess[i] = '?' # mark guess with different
                           # character to avoid match

    # count inexact matches
    white_pins = 0
    for i in range(len(guess)):
        for j in range(len(secret)):
            if guess[i] == secret[j]:
                guess[i] = '?'
                secret[j] = '!'
```

```
                white_pins += 1
    return white_pins
```

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.

Solution:

```
COLORS = 'RGBYPO' # red, green, blue, yellow, purple, orange

def is_valid(guess):
    '''(str) -> bool
    Returns True when guess is valid, meaning
    that it is 4 characters long and it consists
    only of characters in COLORS.
    '''
    if len(guess) != 4:
        return False
    for ch in guess:
        if ch not in COLORS:
            return False
    return True
```

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.

Solution:

```
def prompt_user(guess_num, total_guesses):
    '''(int, int) -> str
    Prompts user for acceptable guess.  Guess
    is acceptable if it is 4 character long
    and it consists only of characters in COLORS.

    Each prompt tells user which guess_num it is
    and the total_guesses they have.

    Returns user's first valid guess.
    '''
    prompt = "Make a guess (" + str(guess_num) + " of " + str(total_guesses) +
```

```
    guess = raw_input(prompt)
    while not is_valid(guess):
        guess = raw_input("Invalid guess. " + prompt)
    return 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'`. Repeat this process four times to build up a secret code.

Solution:

```
COLORS = 'RGBYPO' # red, green, blue, yellow, purple, orange

def generate_code():
    '''() -> str
    Returns 4 character string consisting of
    characters in COLORS, randomly chosen.
    '''
    code = ''
    for _ in range(4):
        idx = random.randint(0, len(COLORS)-1)
        code += COLORS[idx]
    return code
```

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.

Solution: See final solution in `mastermind_final.py` available on moodle.