Program Design

COSC 101: Intro to Computing I
November 8, 2017
DRY

• Don’t
• Repeat
• Yourself
High-Level Tasks
Low-Level Tasks
Even Lower-Level
SOFA

Well-designed functions meet the SOFA criteria:

- Short
- does One thing
- takes Few parameters
- maintains a single level of Abstraction
Top-Down Program Design

A problem solving technique where you approach a large problem by:

1. Starting with general description of problem
2. Break it into several high-level steps
3. Iteratively break the steps into smaller steps until you have steps that are easy to solve.
Linear Approach

walk(bookstore, mcgreory_hall)
walk_southeast(utica_st)
turn_left(broad_st)
turn_left(kendrick_ave)
turn_right(willow_path)
turn_right(academic_dr)
turn_left(academic_dr)
turn_left(academic_dr)
arrive_right(mcgregory_hall)
Hierarchical Approach

walk() →
walk_southeast()  turn_left()  turn_right()  arrive_right()
Testing

1. Test each function independently
2. Test functions that work together
3. Test entire program
4. Test for edge cases and unexpected user actions
def play_again():
    answer = input("Play again? ")
    if answer == 'y':
        play_game()
    print("Well, fine!")

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

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

play_games()
Version #2 is better because it’s structured hierarchically:

As opposed to Version #1 which has a flat structure:
Mastermind Game

Two player roles: Codemaker & Codebreaker

1. Codemaker creates a code using colored pegs and hides them at one end of the board.

2. Codebreaker guesses code by placing row of colored pegs on the board.

3. Codemaker provides feedback by placing small pegs indicating how many pegs were in the right positions and how many colors were correct but in the wrong position.

4. Repeat steps 2 & 3 until code is broken or there are no more rows remaining.
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.

```python
def count_exact(guess, code):
    '''(str, str) -> int
    Compares guess and code and returns number of exact matches (same letter at same index).
    >>> count_exact('RGRY', 'RYGY')
    2
    '''
    count = 0
    for i in range(len(code)):
        if guess[i] == code[i]:
            count += 1
    return count
```
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!

```python
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', 'PGGO')
    1
    '''
```
def count_inexact(secret, guess):
    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
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 \texttt{R}, \texttt{G}, \texttt{B}, \texttt{Y}, \texttt{P}, and \texttt{O}.

COLORS = 'RGBYPO'

```python
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
```

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.

```python
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 ") of "
    prompt = prompt + str(total_guesses) + "): "
    guess = raw_input(prompt)
    while not is_valid(guess):
        guess = raw_input("Invalid guess. " + prompt)
    return guess
```
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.

```python
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
```
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.

```python
def play_game():
    '''() -> NoneType
    Plays a single game of mastermind.
    '''

    secret = generate_code()
    num_guesses = 0
    black_pins = 0

    while num_guesses < MAX_GUESSES and black_pins < 4:
        guess = prompt_user(num_guesses+1, MAX_GUESSES)
        black_pins = count_exact(secret, guess)
        white_pins = count_inexact(secret, guess)
        print "You have", black_pins, "black pins and",
        print white_pins, "white pins."
        num_guesses += 1

    if black_pins == 4:
        print "You win!"
    else:
        print "Nice try, the secret was", secret
```