CS106AP Practice Midterm Exam Solutions

Summer 2019

Trace: iddle?

- What is returned by the function call:

```
"mior"
```

- What is each function doing? (a brief description here is fine!)
    - The **`griddle`** function does nothing. Since it does not return a value, this function has no impact on the execution of the program.
    - The **`fiddle`** function takes in a target string **`t`** and a search string **`s`** and returns the index of the last occurence of **`s`** in **`t`**
    - The **`riddle`** function takes in a string **`s`** and returns a new string that contains all the unique letters in **`s`**. The string that is returned preserves the original ordering of the letters in **`s`**.

Common Errors

- Mixing up the parameters passed into **`fiddle()`** due to the variables' names
  This trace problem tests your knowledge of variable scope and parameter passing. It's important to remember that variables (baggage tags) only exist inside the functions in which they're created and that similar variable names across functions don't ensure a relationship between those variables.

- Editing s after it's passed into **`griddle()`**
  *Since **`i`** is a string, which is an immutable data type, updating the variable **`i`** inside **`griddle()`** just creates a new string but doesn't change the value that the variable **`s`** points to inside **`riddle()`**. To get the new string, **`griddle()`** would need to return the value, and **`riddle()`** would need to catch the return value and store it back inside **`s`** (or some other variable).*

- Incorrectly updating i within the while loop
  *There's a lot going on inside this while loop! In order to figure out what the **`riddle()`** function is doing, it's useful to trace through an example string line by line to see if you notice a pattern in how it updates the characters. Noticing which lines inside of the while loop fall inside vs. outside the if statement is also helpful for orienting yourself.*

Karel: StrikerKarel

```
def turn_right():
    turn_around()
    turn_left()

def turn_around():
    turn_left()
    turn_left()

def dribble():
    """
    Pre-condition:  Karel is on top of a beeper, and its front is clear.
    Post-condition: Karel is on top of a beeper, having moved once in the
    direction it was originally facing
    """
    pick_beeper()
    move()
    put_beeper()

def find_wall_opening():
    """
    Pre-condition: Karel is facing east in some avenue and its front is
    blocked.
    Post-condition:  Karel is facing east in the same avenue and its front is
    clear.
    """
    turn_left()
    # go as far North as possible
    while front_is_clear():
        dribble()
    turn_around()
    # go South until you reach an opening in the wall
    while not left_is_clear():
        dribble()
    turn_left()

def main():
    """
    Karel stops upon reaching the goal, which is the only situation where it
    can be blocked to the North, South, and East
    """
    while left_is_clear() or right_is_clear() or front_is_clear():
        if not front_is_clear():
            find_wall_opening()
        dribble()
```

Images: Three is better than one

a) Triplicating a single image

```
def triplicate(image_filename):
    image = SimpleImage(image_filename)
    # output image should be three times as wide as input image
    out = SimpleImage.blank(width=3*image.width, height=image.height)
    for y in range(image.height):
        for x in range(image.width):
             source_pixel = image.get_pixel(x,y)

             left_third_pixel = out.get_pixel(x, image.height - y - 1)
             middle_third_pixel = out.get_pixel(x + image.width, y)
             right_third_pixel = out.get_pixel(out.width - x - 1, y)

             # Populate the upside down left third
             left_third_pixel.red = source_pixel.red
             left_third_pixel.blue = source_pixel.blue
             left_third_pixel.green = source_pixel.green
             # Populate the original middle third
             middle_third_pixel.red = source_pixel.red
             middle_third_pixel.blue = source_pixel.blue
             middle_third_pixel.green = source_pixel.green

             # Populate the reflected right third
             right_third_pixel.red = source_pixel.red
             right_third_pixel.blue = source_pixel.blue
             right_third_pixel.green = source_pixel.green

    return out
```

b) Triplicating all images

```
def process_all_images(filename):
    with open(filename, 'r') as f:
        for line in f:
            image_filenames = line.split()
            for image_filename in image_filenames:
                img = triplicate(image_filename + '.png')
                # we don't specify that you have to call .show() on
                # the image, but it might make sense to do so here
                img.show()
```

Dictionaries: Counting characters

```
def frequency_analysis(s):
    counts = {}
    for ch in s:
        letter = ch.lower()
        if letter not in counts:
             counts[letter] = 0
        counts[letter] += 1

    for letter in sorted(counts.keys()):
        print(letter, counts[letter])
```

Console program: Find range

```
SENTINEL = 0

def find_range():
    print(">>> This program prints the largest and smallest numbers.")

    current_num = int(input(">>> Enter an integer or " + str(SENTINEL) + " to
stop: "))

    smallest = current_num
    largest = current_num

    while current_num != SENTINEL:
        if current_num < smallest:
            smallest = current_num
        if current_num > largest:
            largest = current_num

        current_num = int(input(">>> Enter an integer or " + str(SENTINEL) + "
to stop: "))

    print("Smallest: " + str(smallest))
    print("Largest: " + str(largest))
```