Nicholas Bowman, Sonja Johnson-Yu, Kylie Jue **Section #2**
CS106AP **July 2, 2019**

Section #2: Solutions

1. Practice with Expressions and Variables

The expressions evaluate to the following values:

- `5 + 3 / 2 - 4` → `2.5`
- `5 + 3 // 2 - 4` → `2`
- `1 * 6 + (5 + 3) % 3` → `8`
- `'abc' + str(1) + str(2)` → `abc12`
- `'abc' + str(1 + 2)` → `abc3`

2. Buggy Bill

In the buggy version of the code, the program outputted the following lines:

- Your total before tip is: $100.
- Your final price is: $125.0.

To fix the code, you need to make two adjustments:

- The call to `calculate_bill()` in `main()` should be

  ```
  calculate_bill(num_pizzas, num_salads)
  ```

  In the original code, the two parameters were swapped.
- The `add_tax()` function should return `total` after it has updated it, and `calculate_bill()` should update `total` by assigning the result of the `add_tax()` function call to `total`.

The full correct code is below.

```
# Constants
TAX_RATE = 0.0625
TIP_RATE = 0.25
SALAD_COST = 5
PIZZA_THRESHOLD = 4
LARGE_ORDER_PIZZA_COST = 70
SMALL_ORDER_PIZZA_COST = 20

def add_salad_costs(n):
    """Return the total cost of all n salads"""
    return n * SALAD_COST
```

```
def add_pizza_costs(n):
    """Return the total cost of all n pizzas."""
    if n < PIZZA_THRESHOLD:
        return SMALL_ORDER_PIZZA_COST
    else:
        return LARGE_ORDER_PIZZA_COST

def add_tax(total):
    """Return the total with tax"""
    total *= 1 + TAX_RATE
    return total

def add_tip(total):
    total *= 1 + TIP_RATE
    return total

def calculate_bill(num_pizzas, num_salads):
    total = 0
    total += add_salad_costs(num_salads)
    total += add_pizza_costs(num_pizzas)
    total = add_tax(total)
    print('Your total before tip is: $' + str(total) + '.')
    total = add_tip(total)
    return total

def main():
    num_salads = 4
    num_pizzas = 6
    final_price = calculate_bill(num_pizzas, num_salads)
    print('Your final price is: $' + str(final_price) + '.')
```

3. Mystery Calculation

Recall that a line has an equation of the form $y = ax + b$. This program calculates output values (y) for input values (x) along a line with slope a and intercept b. First, it prompts the user for the line's slope, a, and its intercept, b. Then, the program prompts the user for x values until the user enters the **`SENTINEL`**, or the stop value. The value of **`SENTINEL`** is defined by a constant. For each entered number, it calculates y according to $y = ax + b$ and then prints the y value that corresponds to the user's entered x value. The values for a and b do not change after the user initially enters them, but x and y change with each iteration of the **`while`** loop.

Here is a sample run of the program, with **SENTINEL = -1**. User input is underlined in blue:

```
This program calculates y coordinates for a line.
Enter a value for a: 2
Enter a value for b: 4
Enter a value for x: 5
Result for x = 5 is 14
Enter a value for x: 1
Result for y when x = 1 is 6
Enter x: -1
```

This program should work properly regardless of the value of **SENTINEL**.

4. The Fibonacci Sequence

```
"""
File: Fibonacci.py
--------------------
This program lists the terms in the Fibonacci sequence up to
a constant MAX_TERM_VALUE, which is the largest Fibonacci term
the program will display.
"""

MAX_TERM_VALUE = 10000


def main():
    print('This program lists the Fibonacci sequence.')
    first_term = 0
    second_term = 1
    while first_term < MAX_TERM_VALUE:
        print(first_term)
        next_term = first_term + second_term
        first_term = second_term
        second_term = next_term


if __name__ == '__main__':
    main()
```

5. String Indexing and Slicing Practice

These print statements produce the following output:

- `print(len(s))` → `10`
- `print(s[0])` → `P`
- `print(s[9])` → `e`
- `print(s[3])` → `h`
- `print(s[10])` → `IndexError: string index out of range`
- `print(s[1] + s[3] + s[5] + s[7] + s[9])` → `yhnie`
- `print(s[3] + 100)` → `TypeError: can only concatenate str (not "int") to str`
- `print(s[3] + str(100))` → `h100`

These substrings can be produced using the following slice expressions:

- `'ython'` → `s[1:6]`
- `'Py'` → `s[0:2] or s[:2]`
- `'Tim'` → `s[6:9]`
- `'Time'` → `s[6:10] or s[6:]`
- `'T'` → `s[6:7] or s[6]`
- `'PythonTime'` → `s[0:10] or s[:]`

6. String Building and Analysis

Separation Nation

```
"""
File: Separate.py
--------------------
This program takes in a string from the user and separates the letters
from the numbers before printing them.
"""

def separate_digits_and_letters(s):
    """
    Given a string `s`, this function separates the numbers and letters
    into different strings. Then it returns the concatenation of the
    two strings so that all of the numbers appear first.
    """
    nums = ''
    letters = ''
    for i in range(len(s)):
        current_char = s[i]
        if current_char.isalpha():
            letters += current_char
        if current_char.isdigit():
            nums += current_char
    return nums + letters

def main():
    print('Welcome to Separation Nation, where digits and letters cannot
          peacefully coexist.')
    user_string = input('Please enter a string to separate: ')
    while user_string != '':
        seperated_string = separate_digits_and_letters(user_string)
        print(seperated_string)
        user_string = input('Please enter a string to separate: ')
    print('Goodbye!')

if __name__ == '__main__':
    main()
```

Negative Word Count

```
"""
File: NegativeWordCount.py
--------------------
This program takes in a string from the user and prints how many
words in the user's input string are negative. A word (substring) is
negative if the substring exactly matches ' not ' or is a five letter
string that ends with "'t."
"""

def get_negative_word_count(s):
    """
    Given a string `s`, this function iterates over each five-letter
    substring to count the number of negative words (substrings).
    """
    count = 0
    if len(s) < 5:
        return 0
    for i in range(len(s) - 4):
        current_substring = s[i:i+5]
        if current_substring == ' not ':
            count += 1
        elif current_substring[len(current_substring)-2:] == "'t":
            count += 1
    return count

def main():
    print('This program determines how negative a sentence is.')
    user_string = input('Please enter a string to search through: ')
    while user_string != '':
        num_neg_words = get_negative_word_count(user_string)
        print('This is the number of negative words: ' +
              str(num_neg_words))
        user_string = input('Please enter a string to search through: ')
    print('Goodbye!')

if __name__ == '__main__':
    main()
```