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

Section Handout #2: Python Fundamentals

1. Practice with Expressions

What is the value of the following expressions?

- `5 + 3 / 2 - 4`
- `5 + 3 // 2 - 4`
- `1 * 6 + (5 + 3) % 3`
- `'abc' + str(1) + str(2)`
- `'abc' + str(1 + 2)`

2. Buggy Bill

Similar to the example we've seen in lecture this week, the program below calculates a bill, specifically for Treehouse! But something weird is going on, and customers are getting overcharged…

We should be getting the following two output lines:

- Your total before tip is: $95.625.
- Your final price is: $119.53125.

Trace through the program and answer the following questions:

- What numbers are we getting instead?
- There are a couple of bugs in the code. What are they and how can we fix them?

```
"""
File: TreehouseBill.py
----------------------------
It's your job to figure out what this program does!
"""
# 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

def add_tip(total):
    """Return the total with tip"""
    total *= 1 + TIP_RATE
    return total

def calculate_bill(num_pizzas, num_salads):
    """
    Given the total numbers of pizzas and salads, return
    the total cost of the meal.
    """
    total = 0
    total += add_salad_costs(num_salads)
    total += add_pizza_costs(num_pizzas)
    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_salads, num_pizzas)
    print('Your final price is: $' + str(final_price) + '.')
```

3. Mystery Calculation

The interactive console program below performs a type of calculation that might seem familiar. Examine the code and answer the following questions:

- What is the role of the **SENTINEL** variable?
- How do each of the four variables – **a**, **b**, **x**, and **y** – change over time?
- Overall, what task does this program accomplish?

```
"""
File: MysteryCalculation.py
---------------------------
It's your job to figure out what this program does!
"""

def main():
    SENTINEL = -1
    a = int(input('Enter a value for a: '))
    b = int(input('Enter a value for b: '))
    x = int(input('Enter a value for x: '))
    while x != SENTINEL:
        y = a * x + b
        print('Result for y when x = ' + str(x) + ' is ' + str(y))
        x = int(input('Enter a value for x: '))
```

4. The Fibonacci Sequence

In the 13th century, the Italian mathematician Leonardo Fibonacci – as a way to explain the geometric growth of a population of rabbits – devised a mathematical sequence that now bears his name. The first two terms in this sequence, **Fib**(0) and **Fib**(1), are 0 and 1, and every subsequent term is the sum of the preceding two. Thus, the first several terms in the Fibonacci sequence look like this:

Fib(0)	=	0	
Fib(1)	=	1	
Fib(2)	=	1	(0 + 1)
Fib(3)	=	2	(1 + 1)
Fib(4)	=	3	(1 + 2)
Fib(5)	=	5	(2 + 3)

Write a program that displays the terms in the Fibonacci sequence, starting with **Fib**(0) and continuing as long as the terms are less than or equal to 10,000. Thus, your program should produce the sample run displayed in Figure 1.

```
This program lists the Fibonacci sequence.
0
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
```

Figure 1: The output of your Fibonacci sequence program

This program should continue as long as the value of the term is less than or equal to the maximum value. To do this, you should use a **`while`** loop with a header line that looks like this:

```
while term <= MAX_TERM_VALUE:
```

Note that the maximum term value is specified using a constant. Your program should work properly regardless of the value of **`MAX_TERM_VALUE`**.

5. String Indexing and Slicing Practice

Given the following line of code:

```
s = 'PythonTime'
```

What would the following expressions print?

- `print(len(s))`
- `print(s[0])`
- `print(s[9])`
- `print(s[3])`
- `print(s[10])`
- `print(s[1] + s[3] + s[5] + s[7] + s[9])`
- `print(s[3] + 100)`
- `print(s[3] + str(100))`

What are the slice expressions that would yield the following substrings of `s`?

- `'ython'`
- `'Py'`
- `'Tim'`
- `'Time'`
- `'T'`
- `'PythonTime'`

Useful hints:

- String indices begin at 0
- Omitting the first index of the slice expression (before the colon) makes the slice start from the very beginning of the string, and omitting the second index of the slice expression (after the colon) makes the slice end at the very end of the string.
- If you get stuck, try to use the Python interpreter to evaluate some of these expressions!

6. String Building and Analysis

The following two problems involve conditional construction of strings, and are very similar to the tasks that you will have to complete on the last two parts of Assignment 2.

- **Separation Nation**

  Write a function

  `separate_digits_and_letters(s)`

  that takes as input a string and returns all of the numbers in the string in their original order of appearance, followed by all of the letters in the string in their original order of appearance. Any other characters present in the original string should be ignored. Then, use this function to build an interactive console program whose output would look like that in Figure 2.

```
Welcome to Separation Nation, where digits and letters cannot peacefully coexist.
Please enter a string to separate: a1b2c3d4
1234abcd
Please enter a string to separate: 2a1dfe3g
213adfeg
Please enter a string to separate: 2a1??dfe!3:g
213adfeg
Please enter a string to separate: 521600
521600
Please enter a string to separate:
Goodbye!
```

Figure 2: A sample output for your Separation Nation program

- **Negative Word Count**

  Write a function

  `get_negative_word_count(s)`

  that takes as input a string **`s`** and returns the count of all negative words (substrings) in **`s`**. A substring is negative if **one of these conditions** is true:

  - the substring is equal to the string ' not ' (Note the spaces around "not"!)
  - the substring is exactly 5 letters and ends with "'t" (e.g. "can't" or "won't")

  Then use this function to build an interactive console program whose output looks like the output in Figure 3 (user input shown in bold and green).

```
This program determines how negative a sentence is.
Please enter a string to search through: I am not happy.
This is the number of negative words: 1
Please enter a string to search through: Take note of how pleasant this experience was.
This is the number of negative words: 0
Please enter a string to search through: This wasn't a good experience and I did not enjoy myself.
This is the number of negative words: 1
Please enter a string to search through:
Goodbye!
```

Figure 3: A sample output for your Negative Word Count program