CSI 1401 Week 5 Resources

by Allen Yan

This weeks, CSI 1401 will finish the rest of the chapter 4 and host the first exam. The exam content will cover everything from chapter 1-3 and 4.1-4.5. My resources from previous weeks may provide some help for exam review. You can find them at https://www.baylor.edu/support_programs/index.php?id=967950.

Additionally, , I will be leading a group tutoring session on CSI 1401 every Wednesday from 7:00 pm to 8:00 pm (central time). The session will be conducted online via Microsoft Meetings, where I will be available to provide interactive help to students. If the above time window does not work for you, or if you need additional help, Baylor's Tutoring Center also provides 1-on-1 online tutoring appointments. For more information on signing up for group tutoring or individual tutoring, please visit https://www.baylor.edu/tutoring.

Week 5 Important Topics

Chapter 4 – Branching (Continued)

- Order of evaluation
- Code blocks and indentation
- Conditional expression

1. Order of evaluation

If you have taken math class, the word "order of operation" may sound familiar to you. Since programming oftentimes involves mathematical operations, Python also features its own "order of operations" when evaluating mathematical expressions. In Python, mathematical operators are evaluated in the order of something called the *precedence rule*. The precedence rule for the operators covered in class is listed in the table below by order of importance. Operators higher on the table will be prioritized during evaluation.

Operator	**Description**
()	Parentheses
* / %	Multiply, divide, modulus
+ -	Add, subtract
< <= > >= == !=	Relational & inequality operators
not	logical NOT
and	logical AND
or	logical OR

You may find that most of the precedence rules are fairly intuitive, since they tend to follow the same precedence convention taught in math classes. The ones worth paying attention to are the

operators used in Boolean expressions, since it is not as obvious that AND would be prioritized over OR during evaluation. It is likely that precedence rule will show up either as a problem or part of a problem on the upcoming exam and future exams, it may be a good idea to get some practice. Consider the example expression below:

```
y + 19 == x / 10 and (z or a)
```

By the precedence rule stated above, the parentheses will have the highest precedence. So, the contents inside of it will be evaluated first, which is z or a. Note that normally the logical OR would be the last thing to be considered because of its low precedence. But the parentheses are used to increase its precedence. Next, the arithmetic operators would be evaluated, so x/10 will be evaluated next, followed by y+19. We then move down the hierarchy and evaluate the == operator, and lastly finish off with evaluating the logical AND.

Operator precedence can quickly get confusing for long expressions, especially those containing high number of Boolean operators. For the sake of readability, I encourage using a lot of parentheses when programming expressions with a lot of Boolean operations, even in cases where the expression evaluates correctly without them. Nevertheless, knowing the precedence rule by heart is very useful.

2. Code blocks and indentation

A *code block* is a section of code grouped together. Code blocks are defined by indentation. The amount of indentation used for each level can be arbitrary, but typically 4 columns are used per indentation level. Python is unique from many other programming languages in that it is strict in checking indentation. Many languages, such as C++, do not care whether you write an entire block of code properly indented or just throw everything in a single giant line – it will still execute properly – but not Python. Python will throw an IndentationError even if one line is mis-indented.

One common pitfall in Python indentation is the mix between tabs and spaces. Many code editors consider tabs to be equivalent to 3 or 4 spaces, but in Python a tab is only equivalent to another tab. It is good practice to only use spaces when indenting Python code, and to set text editors to output a certain number of spaces instead when using the tab key.

Below is a demonstration of code blocks borrowed from the course's zyBooks, with some additional comments added by me:

```
# First code block has no indentation

model = input('Enter car model: ')
year = int(input('Enter year of car manufacture: '))

antique = False
domestic = False

if year < 1970:
    # New code block has indentation of 4 columns
    antique = True

# Back to code block 0
```

```
if model in ['Ford', 'Chevrolet', 'Dodge']:
  # New code block has indentation of 2 columns
  # Any amount of indentation > 0 is OK.
  # PyCharm will complain here, however, because it does not like
indentation not being multiples of 4.
  # But the code is syntactically legal.
  domestic = True

# Back to code block 0

if antique:
    # New code block has indentation of 4 columns
    if domestic:
        # New block has 4 additional columns (8 total)
        print('My own model-T still runs like a charm...')
```

3. Conditional expression

If you have used a lot of if statements in your code, you may find them to be rather verbose, with each new statement requiring a new code block. Traditional if statements are good for readability when there is a lot code in the code blocks, but if each block only has one line of code, it can appear kind of unnecessary. Conditional expression offers a nifty short-hand to make small if statements concise. They take on the following format:

```
expr_when_true if condition else expr_when_false
```

Conditional expression can reduce short if statements with only one else block down to a single line. It uses three operands: the expression when condition is true, the condition itself, and the expression when condition is false. This is sometimes called a ternary operation. Conditional expressions are concise, but can make the code difficult to read when overused, so use them sparingly. See below for a comparison between a traditional if-else statement and its logical equivalent in conditional expression.

```
# Traditional if-else statement
if x % 2 == 0:
    y = 1
else:
    y = 0

# Conditional expression
y = 1 if (x % 2 == 0) else 0
```

Useful Resources

- Basic Python Tutorial on GeeksforGeeks:
  https://www.geeksforgeeks.org/python-programming-language/
    - This page provides links to detailed explanations to many entry-level Python concepts along with examples. I encourage taking a look at it if the examples in your textbook were not clear enough.
- Official Python Documentation:
  https://docs.python.org/3/
    - This may be a bit heavy-handed for a beginner-level programmer since the official Python documentation is very thorough and technical. However, learning how to read official documentations is crucial to becoming a good programmer, because the official documentation contains information on everything you can find about Python elsewhere and more. Therefore, I encourage slowly easing yourself into learning how to read the official documentation.
- Resources of Previous Weeks:
  https://www.baylor.edu/support_programs/index.php?id=967950