CSI 1401 Week 4 Resources

by Allen Yan

This weeks, CSI 1401 will introduce the concept of branching in programming (chapter 4). Branching is one of the most commonly-used form of logic in programming and will enable the user to write much more meaningful programs. Two weeks of class time will be spent on chapter 4.

As a reminder, 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 4 Important Topics

Chapter 4 - Types

- If-else branches
- Equality and relational operators
- Boolean operators and expressions
- Membership and identity operators

1. If-else branches

The most basic type of branching in Python is the if-else branches. Like the name implies, if-else branches allow the program to *branch out* do one thing *if* a certain condition is met, but branch to do another thing otherwise. For example, consider the example program below which tells whether a number is even or odd:

```
my_num = int(input('Enter a number: '))
if my_num % 2 == 0:
    print("Number is even.")
else:
    print("Number is odd.")
```

If your program requires a more sophisticated level of branching than the simple one-or-the-other logic used in the example program above, you may elect to use else-if statements between your initial if and else statements. Else-if statements allow the program to evaluate additional conditions you set before defaulting to the else statement. You may add as many else-if statements as you need, and Python will never complain about it (although adding too many of them will make the code hard to read). For example, the "fruit detection" program below demonstrates else-if statements, note that else-if statements are simplified as "elif" in Python:

```
my_fruit = input('Enter a fruit: ')
if my_fruit == "Apple":
    print("Apples are delicious!")
elif my_fruit == "Banana":
    print("Bananas are great!")
elif my_fruit == "Strawberry":
    print("Strawberries are yummy!")
elif my_fruit == "Grape":
    print("Grapes are nutritious!")
else:
    print("Sorry, I don't know what fruit that is.")
```

Keep in mind that the else statement at the very end is always guaranteed to execute if none of the above if and elif statements are executed. Sometimes you may not want broad behaviors like that in your program. In which case, you can leave out the entire else statement and have your code consist of only if and/or elif statements, it is syntactically okay to do that.

You can also put if statements inside other if or elif statements to create *nested if-else statements*, where the inner if statements are only evaluated if the condition for the outer if statement is met. See the following example code:

```
my_fruit = input('Enter a fruit: ')
if my_fruit == "Apple":
    print("Apples are delicious!")
elif my_fruit == "Banana":
    num = int(input('How many? '))
    if num == 1:
        print("1 banana is not enough!")
    elif num == 2:
        print("2 banana is just right!")
    elif num == 3:
        print("3 banana is too much!")
```

2. Equality and relational operators

If statements work by checking whether the expression after the if or elif keyword evaluates to a Boolean true or false. Boolean is a type of variable that has only two valid values: true and false. There are many ways to provide if statements with Boolean values, but the most common way is to use the equality and relational operators. The table below provides some basic equality and relational operators:

Operator	**Example**	**Returns**
`==`	`a == b`	True if a is equal to b False otherwise
`!=`	`a != b`	True if a is NOT equal to b False otherwise
`<`	`a < b`	True if a is less than b False otherwise

>	a > b	True if a is greater than b False otherwise
<=	a <= b	True if a is less than or equal to b False otherwise
>=	a >= b	True if a is greater than or equal to b False otherwise

You can use equality and relational operators on integers, characters, strings, and floating-point variable types. You should avoid using the equality operators (== and !=) on floating-point types because they have imprecise representation (discussed at a later time), so two floats that appear the same to you may actually have different values underneath the code. When relational operators are used on strings, the individual characters are compared by their ASCII values.

3. Boolean operators and expressions

Boolean operators take Boolean operands and evaluates to a Boolean. Boolean expressions are expressions using Boolean operators. See the table below for some typical Boolean operators in Python:

Operator	**Example**	**Returns**
and	a and b	True only if both a and b are true
or	a or b	True if either a or b, or both are true
not	not a	True if a is false False if a is true

Boolean expressions allow you to fit complex logic evaluations into single lines of code, which is very useful for making your code concise and readable. See the following code segment for example, and imagine how many more lines of code will be needed to make the same program if Boolean expression is not used:

```
a = int(input('Enter a: '))
b = int(input('Enter b: '))
if (a+b)*2 > 10 or (a-b)*2 < -10:
    print("Your numbers are within my range!")
else:
    print("Your numbers are outside my range!")
```

4. Membership and identity operators

Python comes with some rather nifty membership operators that can save you a lot of work when looking for things in containers: the *in* and *not in* operators. As the name implies, the in operator evaluates to Boolean true if the left operand can be found inside the right operand (the container), and the not in operator behaves like the opposite. See the following code segment for an example.

```
cs_prof = ['Fry', 'Booth', 'Aars']
name = input("Enter a name: ")
if name in cs_prof:
    print(name, "is a CS professor")
```

```
else:
    print(name, "is not a CS professor")
```

Python also features identity operators: the *is* and *is not*. They are useful for checking whether two variables are bound to the same object. Note that identity operators are not the same as equality operators, the former evaluates whether two variables refer to the same object underneath the code, while the latter only checks whether two variables possess the same value. See code below for an example that differentiates between the two types of operators.

```
a = [1, 2, 3]
b = [1, 2, 3]
c = a

# will print True
print(a is c)
# will return False, even though a and b have same value
print(a is b)
# will return True
print(a == b)
```

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.
- Python Operators Demo: https://www.w3schools.com/python/python_operators.asp
    - This page contains demonstration code for all of the new Python operators covered this week and more. You can also mess around with the code and see what changes.