Lecture 6 — Decisions

Initial Example:

- Suppose we have a height measurements for two people, Chris and Sandy. We have the tools to write a program that determines which height measurement is greater:

```
chris_height = float(input("Enter Chris's height (in cm): "))
sandy_height = float(input("Enter Sandy's height (in cm): "))
print("The greater height is", max(chris_height,sandy_height))
```

- But, we don't have the tools yet to *decide* who has the greater height. For this we need *if statements*:

```
chris_height = float(input("Enter Chris's height (in cm): "))
sandy_height = float(input("Enter Sandy's height (in cm): "))
if chris_height > sandy_height:
    print("Chris is taller")
else:
    print("Sandy is taller")
```

- This is the first of many lectures exploring logic, if statements, and decision making.

Overview — Logic and Decision Making

- Boolean logic
- Use in decision making
- Use in branching and alternatives

Reading: Chapter 5 of *Practical Programming*. We will not cover all of this today, and will return to the rest of this chapter later in the semester.

Part 1: Boolean Values

- Yet another type
- Values are only `True` and `False`

- We'll see a series of operations that either produce or use boolean values, including relational operators such as `<`, `<=`, etc. and logical operations such as `and` and `or`.
- We can assign them to variables, as in,

```
x = True
```

  although we will not explore this much during the current lecture.

Relational Operators — Less Than and Greater Than

- Comparisons between values, perhaps values associated with variables, to produce a boolean outcome.
- For numerical values, `<`, `<=`, `>`, `>=` are straightforward:

```
>>> x = 17
>>> y = 15.1
>>> x < y
False
>>> x <= y
False
>>> x <= 17
True
>>> y < x
True
```

- The comparison operators `<`, `<=`, `>`, `>=` may also be used for strings but the results are sometimes a bit surprising:

```
>>> s1 = 'art'
>>> s2 = 'Art'
>>> s3 = 'Music'
>>> s4 = 'music'
>>> s1 < s2
False
>>> s2 < s3
True
>>> s2 < s4
True
>>> s1 < s3
False
```

- With strings, the ordering is what's called *lexicographic* rather than purely alphabetical order:
  - All capital letters come before small letters, so strict alphabetical ordering can only be ensured when there is no mixing of caps and smalls.

Relational Operators: Equality and Inequality

- Testing if two values are equal uses the combined, double-equal symbol `==` rather than the single `=`, which is reserved for assignment.
    - Getting accustomed to this convention requires practice, and it is a common source of mistakes
- Inequality is indicated by `!=`.
- We will play with a few examples in class.

Part 1 Exercises

We will stop here and give students a chance to work on the first lecture exercise.

Part 2: if Statements

- General form of what we saw in the example we explored at the start of lecture:

```
if condition:
   block1
else:
   block2
```

  where
    - `condition` is the result of a logical expression (a boolean), such as the result of computing the value of a relational operation
    - `block1` is Python code executed when the condition is `True`
    - `block2` is Python code executed when the condition is `False`
- All statements in `block1` and `block2` must be indented the same number of spaces
- The `block` continues until the indentation stops, and returns to the same level of indentation as the statement starting with `if`
- The `else:` and `block2` are optional, as the following example shows.

Example: Heights of Siblings

- Here is a more extensive version of our initial example, implemented using two consecutive `if` statements and not using an `else`:

```
name1 = "Dale"
print("Enter the height of", name1, "in cm ==> ", end='')
height1 = int(input())

name2 = "Erin"
print("Enter the height of", name2, "in cm ==> ", end='')
height2 = int(input())

if height1 < height2:
    print(name2, "is taller")
    max_height = height2

if height1 >= height2:
    print(name1, "is taller")
    max_height = height1

print("The max height is", max_height)
```

- Writing two separate `if` statements like this, while good as an illustration, is not a good idea in practice. We need to read the code to understand that the two `if` statements produce mutually exclusive results. Instead we should use `else`

```
name1 = "Dale"
height1 = int(input("Enter the height of " + name1 + " in cm ==> "))

name2 = "Erin"
height2 = int(input("Enter the height of " + name2 + " in cm ==> "))

if height1 < height2:
    print(name2, "is taller")
    max_height = height2
else:
    print(name1, "is taller")
    max_height = height1

print("The max height is", max_height)
```

- Notes:
  - The blank lines are added for clarity; they are not required for these programs to have correct syntax.
  - Neither program handles the case of Dale and Erin being the same height. For this we need the next Python construct.

Elif

Recall the kids guessing game where someone thinks of a number and you have to guess it. The only information you are given is that the person who knows the number tells you if your guess is too high, too low, or if you got it correct.

- When we have three or more alternatives to consider we use the if-elif-else structure:

```
if condition1:
    block1
elif condition2:
    block2
else:
    block3
```

- We'll rewrite the height example to use `elif` to handle the case of Dale and Erin having the same height.
- Notes:
  - You do **NOT** need to have an `else` block.
  - Exactly **one** block of code (block1, block2, block3) is executed! Don't forget this!
  - If we leave off the `else:` and block3, then it is possible that none of the blocks are executed.
  - You can use multiple `elif` conditions and blocks.

Part 3: More Complex Boolean Expressions, Starting with and

Consider the following piece of Python code that outputs a message if it was above freezing both yesterday and today

```
cel_today = 12
cel_yesterday = -1
if cel_today > 0 and cel_yesterday > 0:
    print("It was above freezing both yesterday and today.")
```

- A boolean expression involving `and` is `True` if and only if **both** the relational operations produce the value `True`

More Complex Boolean Expressions — or

Consider the following:

```
cel_today = 12
cel_yesterday = -1
if cel_today > 0 or cel_yesterday > 0:
    print("It has been above freezing in the last two days.")
```

- A boolean expression involving `or` is `True` if ANY of the following occurs:
  - the left relational expression is `True`,
  - the right relational expression is `True`,
  - **both** the left and right relational expression are `True`.
- This is called the *inclusive or* and it is somewhat different from common use of the word *or* in English.
- For examples, in the sentence

  > You may order the pancakes or the omelet.

  usually means you may choose pancakes, or you may choose an omelet, but you may not choose both (unless you pay extra).
  - This is called the *exclusive-or*; it is only used in logic and computer science in very special cases.
  - Hence, `or` always means *inclusive-or*.

Boolean Logic — not

- We can also "logically negate" a boolean expression using `not`.

```
a = 15
b = 20
if not a<b:
    print("a is not less than b")
else:
    print("a is less than b")
```

Final Example - Is a Point Inside a Rectangle

We'll gather all of ideas from class to solve the following example: Suppose the bounds of a rectangle are defined by

```
x0 = 10
x1 = 16
y0 = 32
y1 = 45
```

A point at location x, y is inside the rectangle if

$$x0 < x < x1 \qquad \text{and} \qquad y0 < y < y1.$$

Under what conditions are we outside of the rectangle? Under what conditions are on the boundary?

We will write a program that reads in an x, y coordinate of a point and outputs a message depending on whether the point is inside the rectangle, outside the rectangle, or on the boundary. The final code will be posted on the course website.

Summary and Looking Ahead

- if-else and if-elif-else are tools for making decisions and creating alternative computations and results
- The conditional tests involve relationship operators and logical operators
    - Be careful of the distinction between `=` and `==`
- In Lecture 11 we will review boolean logic and discuss more complex if structures.