Decision structures in Python

CMSC 104 Section 2
October 14, 2025

Administrative Notes

Quiz 2 is in-class on Thursday

- Open book, open notes, yes you can use your laptop
- You may *not* get outside assistance
- This lecture will *not* be on the quiz

We will do Classwork 5 at the end of this class

Decision Structures

Sometimes you want your code to do one “if” something happens

- E.g., if the user enters something that is not an integer

But you might want your code to do something else if that thing doesn’t happen

- E.g., calculate a number if the user enters something that *is* an integer

And maybe there are more than two choices

- E.g., do one thing if it’s a positive integer; do a second thing if it’s a negative integer or 0; do a third thing if it’s not an integer at all

This is sometimes referred to as “conditional control flow” or “conditional execution.”

if, if-else, and if-elif-else

As always, Python offers multiple ways to solve this problem

- But we’re not going to use “match - case” for example; it’s complex

The way we’re going to use is:

- If you want to do something in one case, but nothing for all other cases, use “if”
- If you want to do something in one case, and something else for all other cases, use “if-else”
- If you have different actions for more than two cases, use “if-elif-else”

A quick reminder about Boolean variables

We talk a lot in this class about int, float, str, and list variables

We have acknowledged a couple of weeks ago that Boolean (“bool”) variables exist

- They have two possible values, True and False
  - Case matters, they have to have an initial upper-case letter and the rest lower-case
  - These are not strings; you never quote them. They are reserved words that have their defined meanings (their Boolean values) whenever they appear in your code

We haven’t actually used them much in this class, up until now

- But we’re about to start using them, ***a lot.***

“if” statements

Syntax:

```
if (boolean-expression):
    #one or more lines of code
```

Where

`boolean-expression` is some Python expression that evaluates to one of the Boolean values, True or False

- It could be a single variable: `if x` where x is either True or False
- It could be a complex expression `if (9 > 17%15) or (x == y) and j:`

Example: see the file ‘october_14_coding.py’

Additional notes

- That colon : at the end of the first line is critical. The “boolean-expression” is everything between the “if” and the colon. All of that gets evaluated by Python at once. If it doesn’t evaluate to True or False, your program gets an error
- If that expression evaluates to True, all the code that is indented underneath the “if” statement gets executed
- If that expression evaluates to False, nothing gets executed. All that code is skipped.
- So just was with “for” loops, indentation is critical.

“if-else” statements

Okay, so you want to move beyond “do one thing, or do nothing”

- You want to do one thing in one case; or something else in another case
  - An example: if the user enters the wrong input, you want to warn the user that the input was incorrect. If the user enters the correct input, go ahead and run the program
  - Another example: you want to do different things based on the age of your user, because laws require restricting certain sites to minors

“if-else” Syntax

Syntax:

```
if (boolean-expression):
    #one or more lines of code
else:
    # one or more lines of code
```

Where:

`boolean-expression` is some Python expression that evaluates to one of the Boolean values, True or False, just as before

Those colons after the boolean expression, and after the word “else” are critical

Example: see the file ‘mod_october_14_coding.py’

“if-else” semantics

If the boolean-expression evaluates to True

- All the lines of code indented under “if” and before “else” are executed
- Nothing indented under “else” will be executed

If the boolean-expression evaluates to False

- All the lines of code indented under “if” and before “else” are skipped. They will NOT be executed.
- All the lines of code indented under “else” will be executed.

As always with Python, indentation is critical. If you indent too many or too few lines of code, it will be a semantic error and your program will produce incorrect results

“if-elif-else” statements

So now you want to move into situations where you have more than two cases to address:

- Example: you want to assign letter grades based on point totals. If student has 895 points or more, the student gets an A. 795 to 894 gets a B; 695 to 794 a C; and so on.

You *could* do that with only “if-else” statements, or even with only “if” statements. But fortunately Python makes it a little easier with “if-elif-else”

“If-elif-else” syntax

Syntax:

```
if (boolean-expression):
     #one or more lines of code
elif (another boolean-expression):
     #one or more lines of code
elif (another boolean-expression):
     #one or more lines of code
     # you can have as many of these “elif’s” as you want.
else:
     # one or more lines of code
```

Where:

Each `boolean-expression` is some Python expression that evaluates to one of the Boolean values, True or False, just as before

Those colons after the boolean expression, and after the word “else” are critical

The semantics:

Python will run through the “if” and then each “elif” in the order you put them. It will go through until one of the boolean expressions evaluates to True.

- Then it will execute the code indented under *that* if or elif statement
- All other code in the structure will be skipped
- It does not matter if other boolean-expressions later in the structure would also evaluate to True. Once the first one has been found, the rest are skipped.

Ideally, you would structure your “elif” boolean-expressions so that only one could possibly be true. But “ideally” doesn’t always occur. If more than one boolean-expression is True, ONLY the code associated with the first will be executed

An illustration of that:

```
# a program illustrating what happens when multiple boolean expressions are True:

if __name__ == "__main__":
    grade = 95
    if grade >90:
        print ("You get an A")
    elif grade > 80:
        print("You get a B")
    elif grade > 70:
        print("You get a C")
    elif grade > 60:
        print ("You get a D")
    else:
        print ("Sorry, you get an F")
```

What’s this “elif” stuff?

It’s short for “else if”

It would make sense to write: “if …else if… else if… else”

But writing “elif” instead of “else if” means 4 keystrokes instead of 7

- And some of us are lazy. :-)