Python Conditionals and Operators

September 14, 2022

Administrative notes

Remember that HW1 is due next Monday, September 19, before midnight

Submit early and submit often!! Only the last version of each file you submit is graded so there’s no harm in submitting partial results early

Conditionals

Code execution:

1. Sequential - execute each line of code, exactly once - no more, no less - then move on to the next line of code
2. Conditional - execute a line of code, once, IF and ONLY IF some condition (or set of conditions) is true. Otherwise, skip this line and do not execute it - all conditions are Boolean - True or False
3. Iterative - execute a line of code, or group of lines of code, multiple times

Conditionals

if, else, and elif

elif is short for “else if” It requires typing four characters instead of 7, and programmers tend to like shortcuts and optimization

Note: Python 3.10 introduces/makes robust the “match…case…” structure to handle conditionals. We will NOT use that in this class. Thus, don’t use this feature of Python 3.10.

Three cases:

1. One option: If a condition is true, do something. Otherwise, do nothing. “If” statement
2. Two options: If a condition is true, do something, Otherwise, do something else. “If….else…” statement
3. More than two options: If a condition is true, do something. Otherwise, check to see if another condition is true; do something else. Otherwise, keep checking conditions until we find one that’s true or we just give up. “If...elif...elif...else…” statement

“If” statement

if {some boolean condition is true}:

    print(“Yay it is true”)

Notes on this:

- Typically boolean conditions are True or False.
  - A trick: any integer value other than 0 is regarded as “True” and any value that’s equivalent to 0 is regarded as False
  - An empty string is False; any non-empty string is True
-
```
if 5:

    print (“5 is true”)
```

Further notes

The boolean condition is everything between “if” and the colon : A colon terminates the condition. It can be as simple or as complex as you want

Indentation matters!! To the python, white space - either tabs or spaces - indicates what’s in the code to be executed. You only have to indent one space, but I’m a believer that you should indent with tabs.

```
If x == 6:
    print (“X is 6”)
    print (“and we’re done”)
```

If x is really equal to 5, this doesn’t print out anything.
This still prints “and we’re done” because that’s not part of the conditional

```
if x == 6:
  print (“X is 6”)
print (“and we’re done”)
```

“if...else …” statement

Ask a student for her major. If she’s a CMSC major, print out “smart choice.” Otherwise print out “there is still time”

```
major = input(“please enter your current major”)

if major == “CMSC”:

    print(“smart choice”)

else:

    print(“there is still time”)
```

Remember that input returns a string, so this comparison is valid

Colon after else, as well!

Indent the code that’s part of the “else” block

“if...else…”

- There must always be at least one line of code under the “if” statement
- You don’t have to have an “else” part, but if you do have an “else” there must be at least one line of code under it.

“if...elif…else”

Input returns a string. This turns it into an integer

Ask a student her age. If it’s less than 18, tell her she’s a minor and faces some restrictions. If it’s between 18 and 21, tell her she’s an adult but still has some limitations. If she’s over 21, tell her she’s legally an adult.

This is called ‘pseudocode’ - an English-language description of what you want to do

```
age = int(input(“Please enter your current age in years.”))
if age < 18:
    print(“Sorry but you are a minor”)
    print(“there are a lot of things you cannot do on your own”)
elif age < 21:
    print(“you are an adult but there are still some things you can’t do”)
else:
    print(“congratulations you are legally an adult”)
```

Notes on “if...elif...else”

- Only one case will have its code executed when the program runs; the rest of the code will be skipped
- There doesn’t have to be an ending “else” condition. If there’s not, we just do nothing and skip all the code
- There must be at least one statement - one line of code - in each case
- Keep your conditions simple. In the previous example, we only got to the ‘elif’ when the age was >= 18. So we didn’t have to put that in the condition - that is, no need to say elif age >= 18 and age < 21: We already know that first part is true.
- You can have as many “elif” statements as you need

Operators

Or, How to do math or create a boolean condition

Python Operators about which we care

Arithmetic - **, -, +, *, /, //, % # / is floating point division; // is integer division; % is modulus

Assignment- =, +=, --, /=, *=

Comparison- ==, >=, <=, !=, >, < -always result in a Boolean value

Logical - and, or, not - always result in a Boolean value

Operator precedence: arithmetic, then comparison, then logical

- Arithmetic: **, then *,/,//, and %; then + and -
- If you have two or more operators at the same precedence, go left to right

Operators

An ***operator*** is a special symbol that is used to carry out some operation on variables or values in Python

- Kind of a circular definition - an operator is used to carry out operations - but we'll clarify that

Types of operators you need to know about now:

- Arithmetic: +, -, *, /, //, %, ** - used to perform arithmetic on ints and floats
- Comparison: ==, !=, >=, >, <=, < - used to compare values
- Assignment: =, +=, -=, *=, /= - used to assign values to variables
- Logical: and, or, not - used to combine Boolean values into another Boolean value

There are other operators that will come up later, but this will get you started

Arithmetic operators

Most of these do the same things you're used to in math class

+	Addition; add two ints or floats. Adding two ints produces an int; anything else produces a float
-	Subtraction; subtract one int or float from another int or float. An int - an int is an int; anything else is a float
*	Multiplication; multiply two ints or floats. Multiplying two ints produces an int; anything else produces a float
/	Floating point division. Divide one int or float by another. Always produces a float
//	Integer division. Divide one int by another. Always produces an int - any "remainder" is truncated
%	Integer modulus. Divide one int by another. Throw away the quotient (the whole number part); this is the remainder.
**	Exponentiation. Raises one int or float to the power of an int or float

Comparison Operators

Again, mostly what you would expected

==	Is the value to the left the same as the value on the right? Are they numerically equal or are they identical strings? = is the assignment operator so you have to use == for comparisons
!=	Not equal
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Assignment Operators

Python lets you take some shortcuts in common situations

=	Assign the value; x = 5 (remember = vs. ==)
+=	Add the number on the right then assign x += 5 is the same as x = x+ 5
-=	Subtract the number on the right from the number on the left then assign x -= 5 is the same as x = x - 5
*=	Multiply the number on the right then assign x *= 5 is the same as x = x* 5
/=	Divide the number on the left by the number on the right then assign x /= 5 is the same as x = x/ 5

Logical Operators: and, or, not (also called “Boolean Operators”)

x	y	x and y
True	True	True
True	False	False
False	True	False
False	False	False

x	y	x or y
True	True	True
True	False	True
False	True	True
False	False	False

X	not X
True	False
False	True

Order of operations

`**`	Highest
`* / % //`	
`- +`	
`<= < > >= != ==`	
`not`	
`and`	
`or`	Lowest

Why operator precedence is important

What's the value of each:

6 * 5.0 + 2 - 5 / 5

(6 * (5.0 + 2) - 5) / 5

6 < 2 or 5 > 1

6 < (2 or 5) > 1 # let's talk about this one