Input & Output; Intro to Operators

September 13, 2021

Administrative Notes

Make sure you do your labs & homeworks

Homework 1 is out; it is due Friday night by midnight - submitted on gl.umbc.edu

About submitting:

- Submit early; submit often
- You can submit an assignment as many times as you want. Only the *last* one you submit will be graded.
  - The way the system works, each time you submit the assignment, your previous submission is deleted and the “current” submission is kept.
- If you wait until just before midnight on Friday to submit for the first time:
  - The server will be unavailable; or your laptop will begin a system upgrade that takes 15 minutes; or the Internet will be out for a while
  - Or your system clock will be off; you’ll think you still have 5 minutes but gl will say no, too late
  - And then you won’t be able to submit at all because the system stops accepting assignments when its clock hits midnight
- SO: when you have a reasonable amount of the homework done, go ahead and submit
  - In the worst case, if you never get anything else done, you’ll get partial credit for the part of the assignment you’ve done

Quick Review from last week:

Types: int, string, float, bool

Variables: symbolic name; has a type; can be changed

- Names can consist of upper and lower case letters, digits, and underscores
- Cannot start with a digit
- CMSC 201 convention: use snake_case for long names, NOT camelCase

Constants: Python doesn’t support; we fake by using ALL CAPS variable names

Literals: use this value, exactly

Input and Output (I/O)

Python’s default is to get all input from the keyboard, and put all output onto the screen

“Print” puts output onto the screen

- You can print variables, literals, or expressions that need to be evaluated
- You can print any type we’ve covered so far

Formatting output:

The basic “Print” statement

Variables and literals

Separate values with commas or plus signs

- Commas means a space is inserted between values
- Plus signs means run strings together with no space

Print statement ends with a newline (\n) unless you explicitly tell it otherwise

Inserting line breaks

The Python “newline” character is \n. When Python encounters “\n” it prints a new line.

```
print(“This will result in one line”, “ being printed”)
print(“This will result in two lines”, “\n”,  “ being printed”)
```

By default, Python prints a new line at the end of every print statement

```
print(“This will result in two lines”)
print(“being printed”)
```

If you don’t want a new line, you can suppress it by using an “end” value

```
print(“This will result in one line”, end=” “)
print(“being printed”)
```

How do you print out a newline character?

Escaping - using \

print(“\\n”)

The tab character is \t. How do you print that out? print(“\\t”)

To print a single quote, print(‘\’’). A double quote is print(‘\”’)

Formatting printed output

Default field lengths when printing:

String: Python takes exactly as many spaces as there are characters in the string

Int: Python takes exactly as many spaces as digits in the integer

Float: Python prints everything to the left of the decimal point, and up to 16 digits after the decimal point

Boolean: Python takes four spaces for True and five spaces for False

Input: the ‘Input’ Statement

The input statement has the syntax:

var_name = input(“Prompt to be printed”)

Whatever the prompt is will be printed; then the program will stop until the user finishes typing in the input

Note: the result of an input statement is ***always*** a string

- If you want it to be something else, you need to cast it (convert it) to a different type

A few notes on the input statement

- The prompt can be a literal or a variable
- But it must be a single string
  - You can’t print multiple strings in a single “input” statement
- You can only bring in one value per input statement
  - If you need multiple values, use multiple input statements
  - Later on in the semester we’ll cover how to bring in multiple values in a single statement

Examples

Fancier Output Formats

Python has a whole *bunch* of different ways to print out results

- There’s always an “improvement” - I’ve taught different things every semester I’ve taught this course

We don’t make it a huge part of this course because it’s not fundamental “computer science” but it is a nice skill to have

This won’t be on the exam!!!

f-strings

Introduced in Python version 3.6

- The “hot new toy” of Python programming
- Similar to str.format, but you can put any Python expression into your statement
- Start with ‘f’ or ‘F’
  - Then include anything you want in a string
  - Variable names get put in curly braces. They will be evaluated at runtime (when the statement is executed) and the actual value will be printed

Multiline f-strings are permitted

```
>>> profession = "comedian"
>>> affiliation = "Monty Python"
>>> message = (
...     f"Hi {name}. "
...     f"You are a {profession}. "
...     f"You were in {affiliation}."
... )
>>> message
  'Hi Eric. You are a comedian. You were in Monty Python.'
```

Setting field sizes

After the variable name or literal value, put a colon (:) and then the minimum number of spaces you want to take up

- Python will ALWAYS expand the field to include the entire string, the entire integer, the entire Boolean, and all the numbers to the left of the decimal point in a float
- For floating point values, the specification is a.b where a is the minimum total number of spaces and b is the maximum number of places to the right of the decimal that will be printed
    - The a value is optional

Let’s spend a lot of time on examples

F-strings: pay attention to

- Justification: if the field size is bigger than the value to be printed:
  - Strings are left-justified
  - Floats and ints are right-justified
  -

Casting types

All input statements result in a string being read in

But what if you need a float or an int?

You can convert by casting variables of one type to another

```
gpa = input("What was your GPA last semester?") #produces a string
actual_gpa = float(gpa)  # converts the string to a float
gpa = float(gpa) # this will work too but can be confusing
```

What if your user types `four point zero` in response to the prompt?
Your program crashes. Typically, that's okay for now.

You can convert any type to any type

str() - cast this variable to a string

int() - cast this variable to integer

float() - cast this variable to a floating point number

bool() - cast this variable to a list

#Make sure the value you’re casting is appropriate to the type to which you’re casting.

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

Thanks, Dr. Johnson

Order of operations

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

Remember, like the noble and majestic honey badger, parentheses don't care about order of operations and will take precedence over everything.

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