Python Lists: what makes code “Pythonic”

CMSC 201 Section 60
September 16, 2024

Administrative Notes

- Homework 1 is due before midnight Tonight!!!
  - Make sure to submit it
- Homework 2 is out
  - It will be due next Monday, September 23, before midnight

Picking up where we left off last Wednesday

String Operators

Relevant for Homework 2:

The “+” operator works like you’d expect for integer and float values

- It’s just standard addition

The “+” operator is used to concatenate strings - it “adds” strings together to produce a longer string

upper() turns a string into the same characters, but all upper case

lower() turns a string into the same characters, but all lower case

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”)
```

For floats: it’s 0.0
0.0 is interpreted as False;
Anything else 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

Now for the new material

What makes a program “Pythonic?”

“Pythonic” means that you’re not just using the syntax of the language, but you’re writing code in the spirit of the language itself.

The first two things that make Python somewhat unique are ***lists*** and ***dictionaries***.

- Dictionaries come later in the semester; for now we’re talking lists.

Variables holding more than one value

All of these types are also immutable - remember what that means? Make sure I discuss this again before the end of the lecture. AND on Wednesday, because it's important.

Up until now, every type we've talked about is a ***scalar*** type. It holds one value.

- int variables contain one integer value
- float variables contain one floating point value
- boolean variables contain one Boolean value
- string variables contain one string value. (Yeah, it may have a lot of characters in it but it's a single value because it's treated as a whole.)

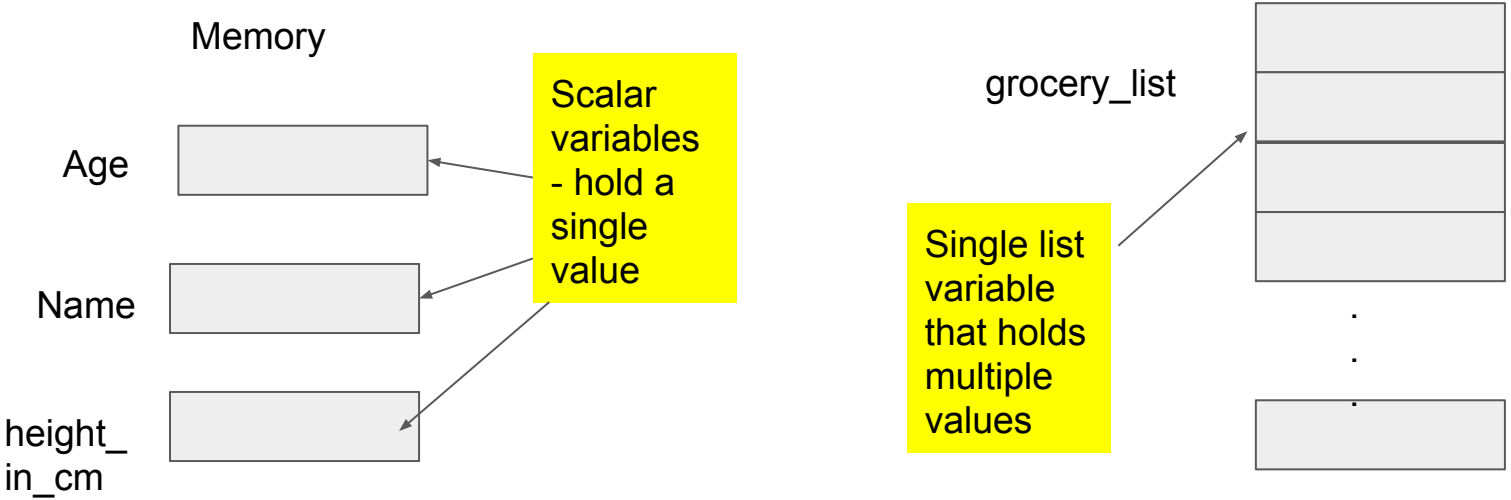
But what if you want to hold more than one value in a variable?

- You'll have 25 grades in this class this semester. You can use 25 different variables to store them, but that's ugly. Why not put them all in one place?

Lists

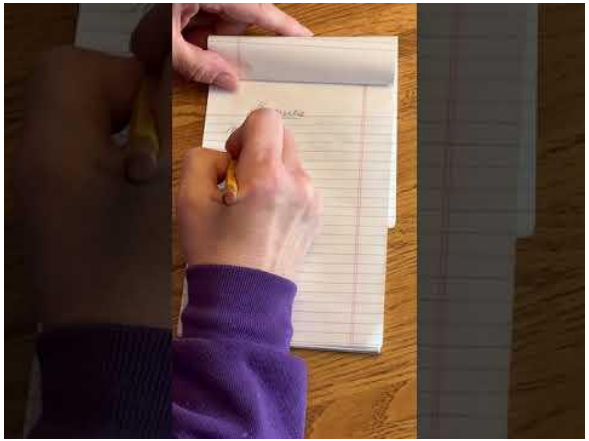
Lists are the fundamental way that Python manages multi-valued variables

- Warning: You might get the impression from lecture and the book that a list is a one-dimensional array. Don't believe that. Arrays are different.



Why?

Make a grocery list - before you knew Computer Science!



Example - my grocery list from this weekend

Milk
Eggs
Cereal
Coffee
Apples
Strawberries
Broccoli
Cucumber
Tomatoes
Green Onions
Half & Half

In this example, everything will be a string

Create a list variable in Python

1. Create an empty list so that we can later add items to it

```
grocery_list = []
```

#Square brackets mean “create an empty list”

2. Create a list with the values already inserted

```
grocery_list = [“Milk”, “Eggs”, “Cereal”, “Coffee”, “Apples”, “Strawberries”, “Broccoli”, “Cucumber”, “Tomatoes”, “Green Onions”]
```

Square brackets mean “create a list.” Elements of the list are separated by commas. Double quotes mean the elements are strings.

We operate on lists using “methods”

“Method” is a term from object-oriented programming that defines an operation that can be performed on a object/variable

Invoked by putting a dot, and then the method name, after the variable name.

Methods for lists:

append - add an element to the end of the list

remove - remove a designated element from the list

Insert - put an element into a designated space in the list

So building the list

```
grocery_list = []   #create an empty list

grocery_list.append(“Milk”)

grocery_list.append(“eggs”)

grocery_list.append(“cereal”)

…

grocery_list.append(“green onions”)
```

Indexing the list

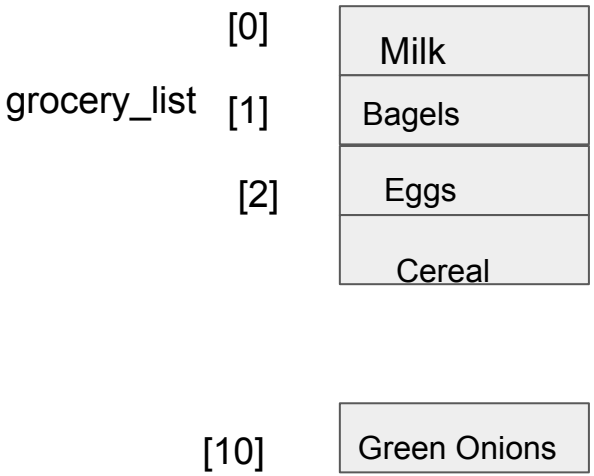
In computer science, we *almost* always start numbering from 0. The first element in any list in Python is [0] - e.g., grocery_list[0] contains “Milk”

Then go up by one. My grocery list has 10 items on it. So the last item, “green onions” is stored in grocery_list [9].

Adding to the list - My wife texts. Can I please pick up a half-dozen bagels, too?

grocery_list.append("bagels")

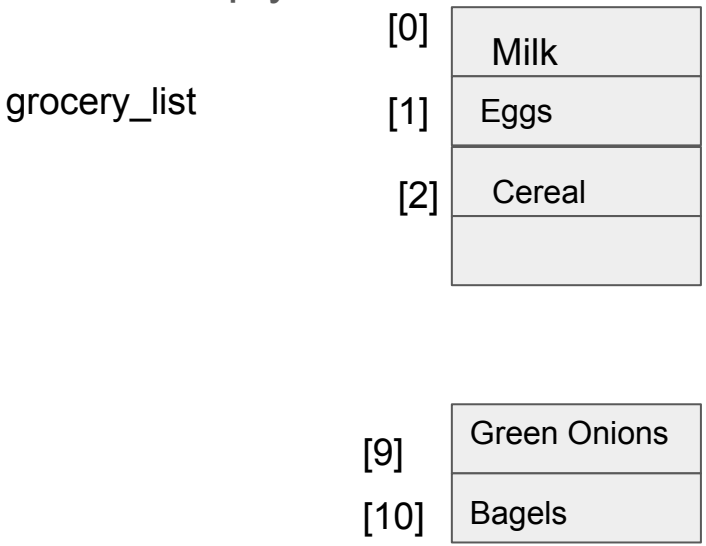
grocery_list.insert(1. "bagels")

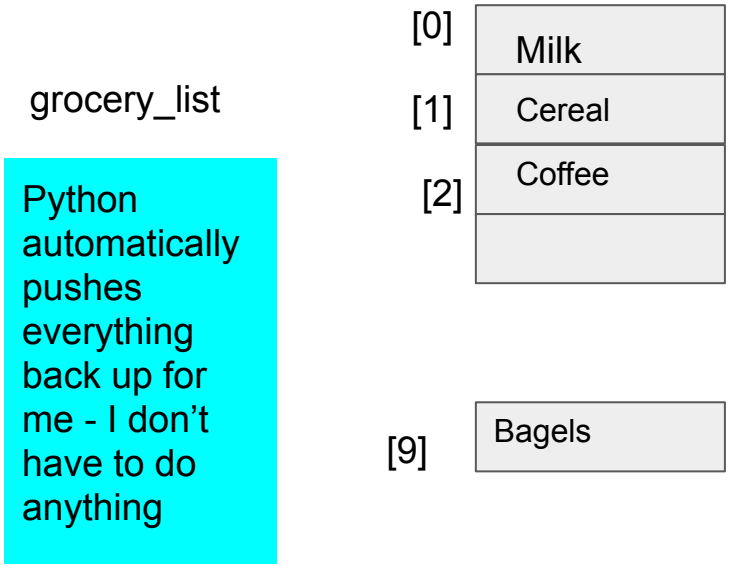


Python automatically pushes everything down for me - I don't have to do anything

Removing an item (by its value)

Suppose I want to remove each item from my list as I put it in the cart. When the list is empty I'm done. Use the .remove method: grocery_list.remove("Eggs")





Removing an element by its index uses a different method - pop

Useful tools to manipulate lists

How long is it? The “len” function.

len(grocery_list)

Note - this gives the total number of elements in the list. So it’s always equal to one more than the last index. Or, the last index is len(grocery_list) - 1.

What happens if you try

grocery_list[len(grocery_list)]

What about grocery_list[len(grocery_list) - 1]

Is a particular value in a list?

The reserved word “in” is useful for this

“Eggs” in grocery_list returns a boolean value

Questions about lists

Do all elements of a list have to be the same type (all ints, all floats, all strings?)

- NO!! Python can sort the types out and manage them. But you'll generally make all of your list elements the same type because otherwise you get into really bad design, really fast

When should I use a list?

- When you have a collection of data elements of the same type that logically go together - *items on my grocery list*!!

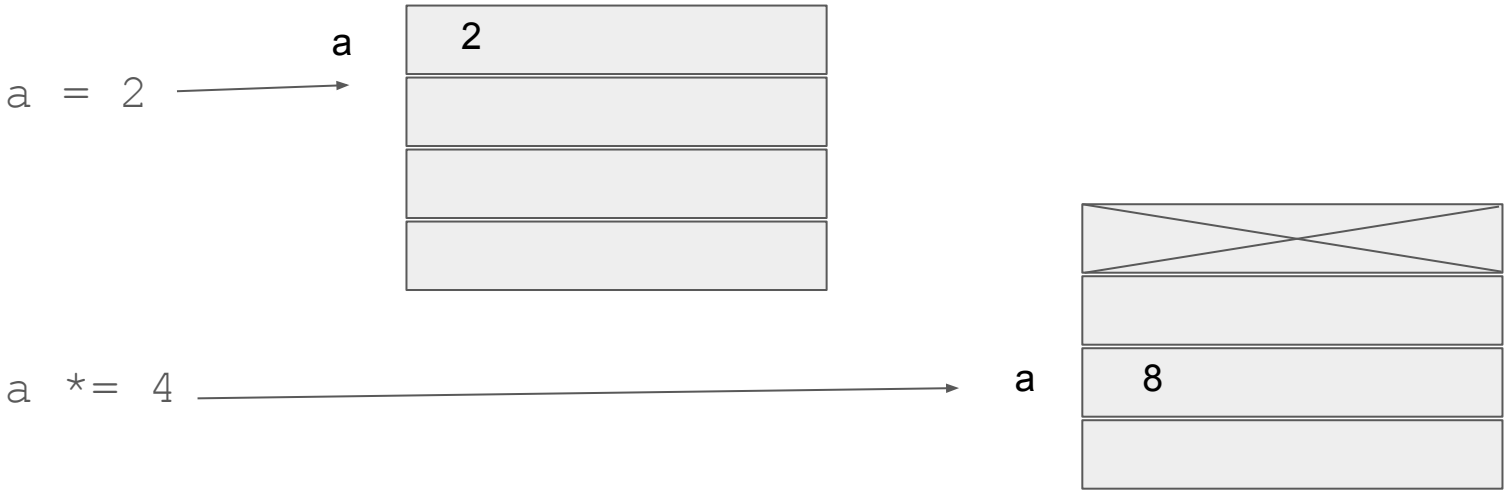
When should I *not* use a list?

- When you have multiple data elements that really don't belong together - *name, birthdate, age, major, height, weight of a student*
  - You would use an array for this, and lists are not arrays!!

Mutability

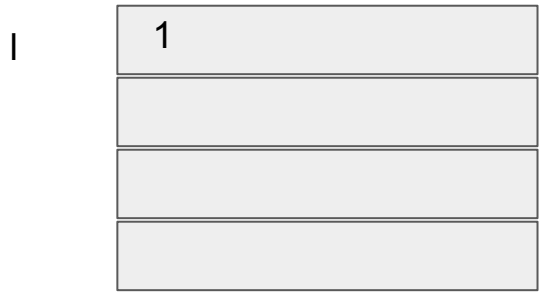
A major way that lists are different from scalar types is that lists are ***mutable***. That means they can be changed after they're declared. Ints, floats, booleans and strings are ***immutable***. They cannot be changed after being assigned a value.

How that works:

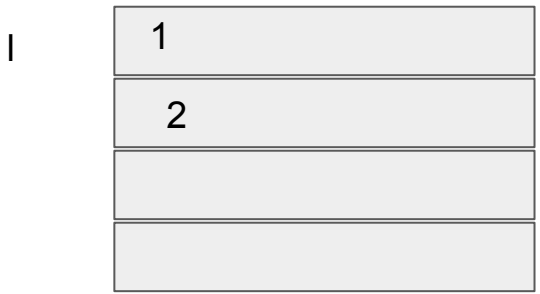


But with a list, it doesn’t work that way

```
l = [1]
```




```
l.append[2]
```



So what?

This has big implications for how Python programs actually work

- We'll be exploring these throughout the semester