Python Lists: what makes code “Pythonic”

September 18, 2023

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 25, before midnight

Picking up where we left off last Wednesday

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 5 != 3 + 2

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 = 30.0 + 2 - 1 = 31.0

(6 * (5.0 + 2) - 5) / 5 = (6 * 7.0 -5) /5 = (42.0 -5)/5 37.0/5 7.4

6 < 2 or 5 > 1 False or True True

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

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

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.)

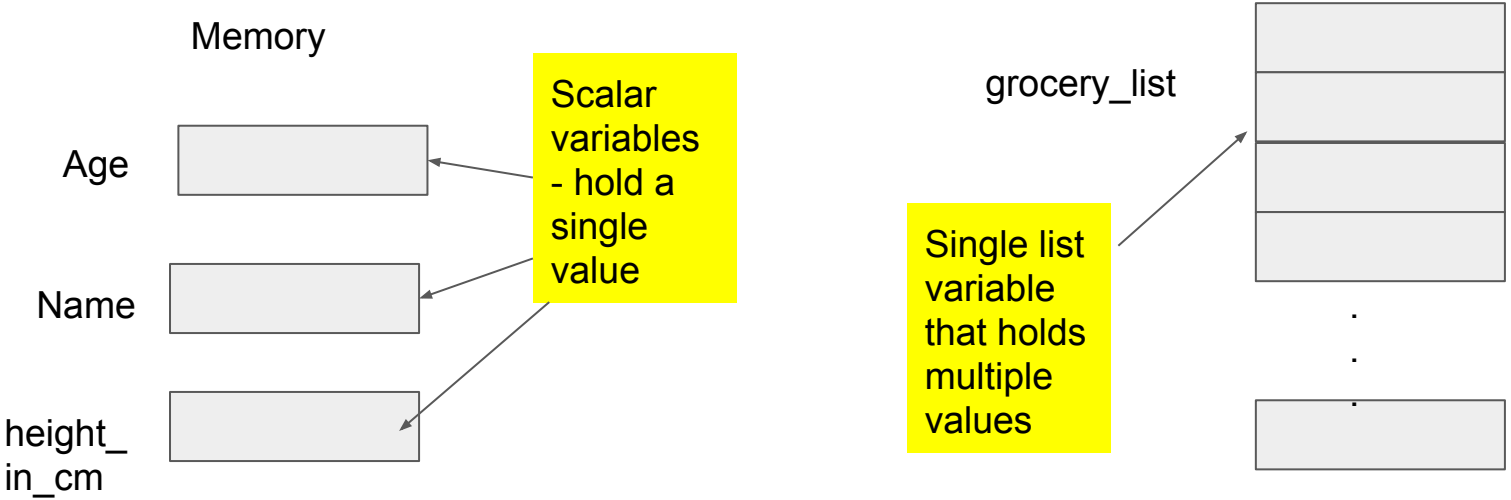
But what if you want to hold more than one value in a variable?

- 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

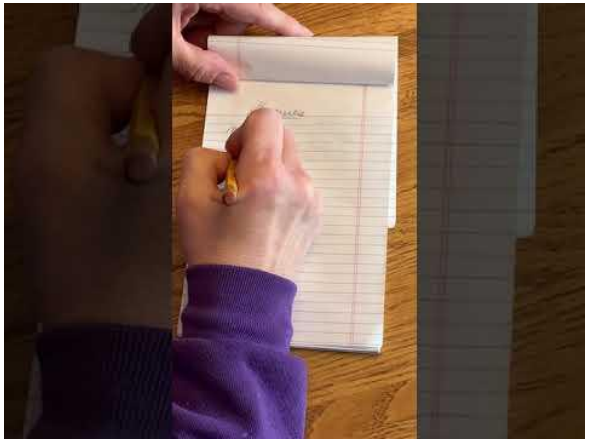
Lists are the fundamental way that Python manages multi-valued variables

- 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

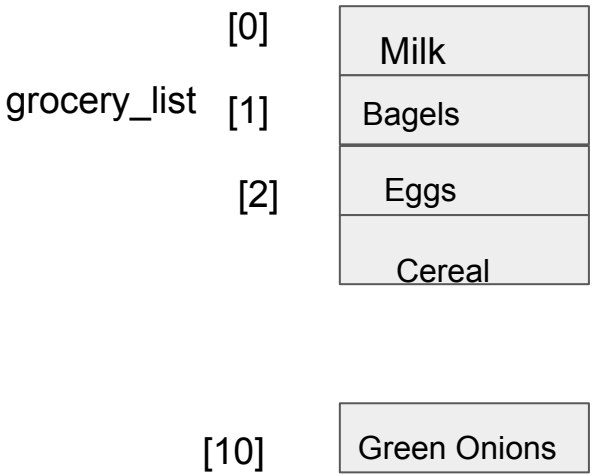
In computer science, we *almost* always start numbering from 0. The first element in any list in Python is [0] - e.g., grocery_list[0] contains “Milk”

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")

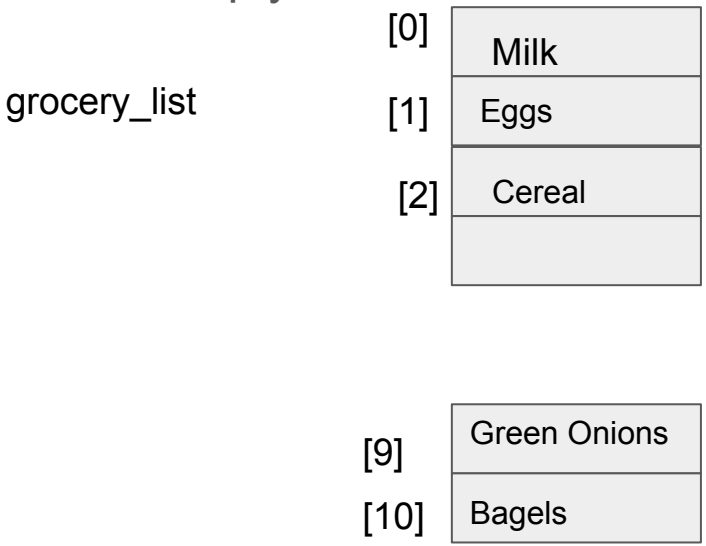
grocery_list.insert(1. "bagels")

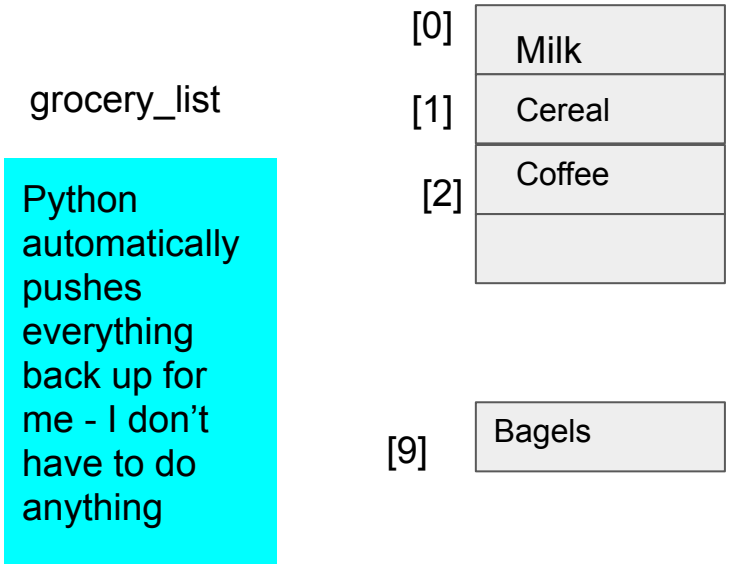


Python automatically pushes everything down for me - I don't have to do anything

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*!!

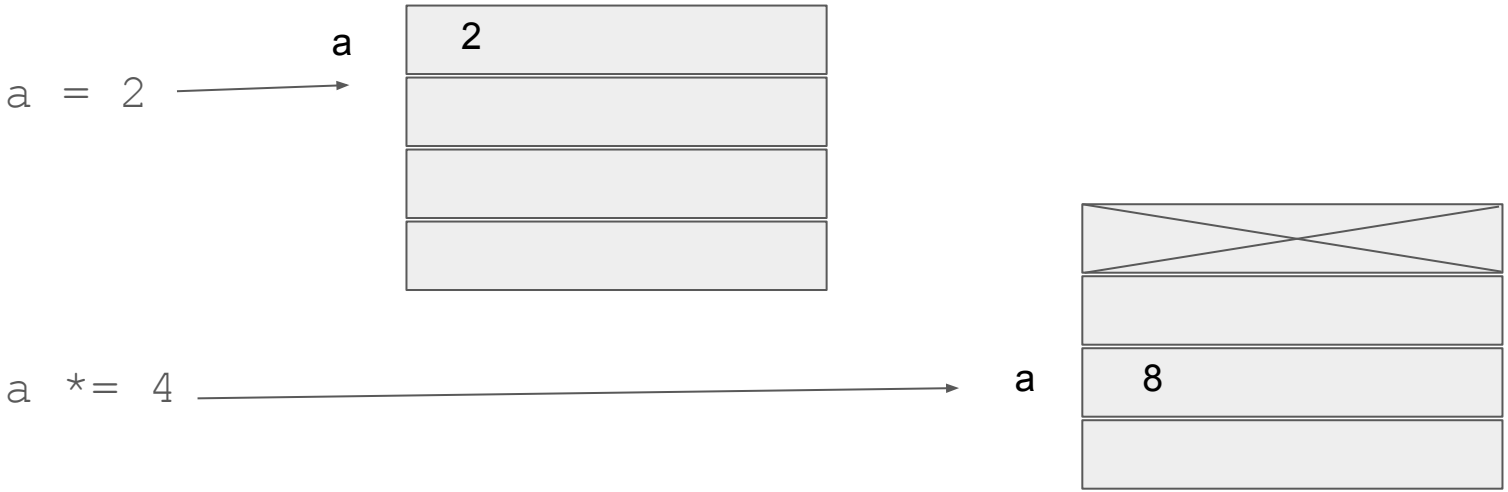
When should I *not* use a list?

- 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

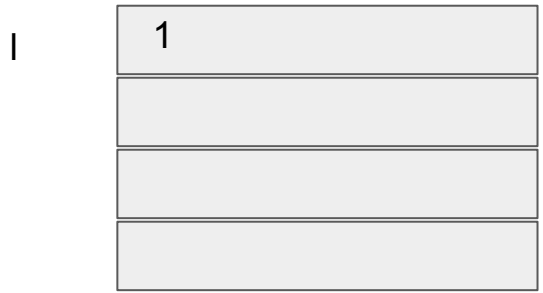
A major way that lists are different from scalar types is that lists are ***mutable***. That means they can be changed after they're declared. Ints, floats, booleans and strings are ***immutable***. They cannot be changed after being assigned a value.

How that works:

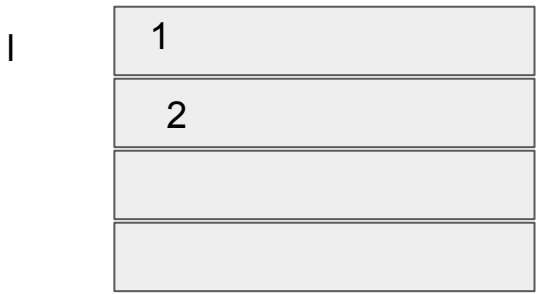


But with a list, it doesn’t work that way

```
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