SCRIPTING AS COOKING: VARIABLES, FUNCTIONS, METHODS

please note: if we want this description to be more correct, we would be talking about objects and classes and object-oriented programming. We will talk about OOP later in the course.

This is our first week of adventuring with Python. We are wrapping our brains around general ideas. It's ok to not be 100% technically accurate right away.

SIMPLE CHICKPEA SALAD



INGREDIENTS:

- 2 cucumbers
- 3 tomatoes
- 1 small red onion
- 1 can (425g) chickpeas
- 2 tablespoons fresh lemon juice
- 1/2 tablespoon minced fresh parsley
- 1 tablespoon extra virgin olive oil
- 1/2 teaspoon kosher salt and pepper, to taste

INSTRUCTIONS:

Dice cucumbers, tomatoes, and onions.

Rinse and drain chickpea.

Combine all the ingredients together.

CAN COMPUTERS COOK?

```
ingredient1 = cucumber
```

ingredient2 = tomato

ingredient3 = onion

ingredient4 = can of chickpea

ingredient5 = lemon

ingredient6 = fresh parsley

ingredient7 = olive oil

ingredient8 = salt

Cooking → manipulating ingredients

Scripting → manipulating variables

RECIPES ARE ALGORITHMS

Recipe: Dice cucumbers, tomatoes, and onions.

pseudocode:

```
# take a cucumber
# wash a cucumber with water
# dice cucumber using a knife
# put diced cucumber in a bowl
# take a tomato
# wash a tomato with water...
```

Recipes → instructions for how to cook, i.e. transform ingredients into a dish

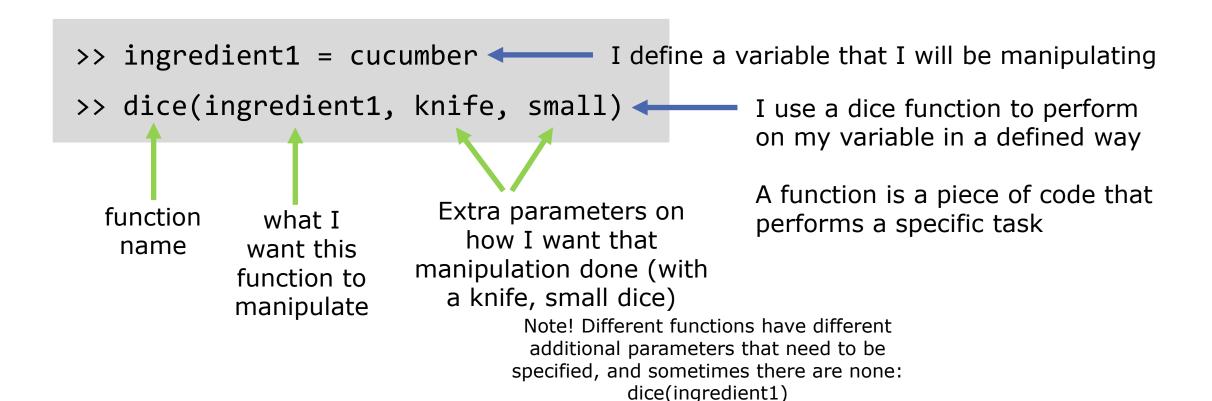
Algorithms → instructions for how to achieve some computational goal, i.e. transform variables into a desired outcome

Depending on how detailed a recipe is, there might be some implied actions – for example, we know that we need to wash vegetables before eating them.

An algorithm must specify all steps, and nothing can be implied.

FAKE CODE EXAMPLE: FUNCTIONS (1)

I want to dice a cucumber:



FAKE CODE EXAMPLE: FUNCTIONS (2)

I want to dice a cucumber, and then I want to add my freshly diced cucumber to a bowl:

```
>> ingredient1 = cucumber
>> container = bowl
>> diced_cucumber = dice(ingredient1, knife, small)
>> add_to_container(diced_cucumber, container)
```

I start with two variables: an ingredient (cucumber) and a container (bowl). I then create a new variable (diced cucumber) by applying the dice function to ingredient1. Then I use the add_to_container function, where I specify what I want added to which container.

FAKE CODE EXAMPLE: METHODS (1)

I know that I can (conceptually) dice a cucumber, and I know that I cannot dice olive oil. Certain functions (manipulations) can only be used on specific types of variables.

Those type-specific functions are called *methods*.

Dicing a cucumber using methods:

- >> ingredient1 = cucumber
- >> ingredient1.diced(knife, small)

. indicates that a method is applied

method name

Extra parameters on how I want that manipulation done (with a knife, small dice)

Note! some methods require no extra parameters, so we leave empty brackets: ingredient1.diced()

FAKE CODE EXAMPLE: METHODS (2)

```
>> ingredient1 = cucumber
>> container = bowl
>> add_to_container(ingredient1.diced(knife, small), container)
```

I start with two variables: an ingredient (cucumber) and a container (bowl). Then I use the add_to_container function, where I specify what I want added (ingredient1 that is diced with a knife to a small size) to which container.

FAKE CODE EXAMPLE: MIXING METHODS

I want a cucumber to be washed and diced!

- >> ingredient1 = cucumber
- >> prepared_ingredient1 = ingredient1.washed().diced(knife, small)