2. Mathematical Foundations

See also the slides that summarize a portion of this content.

Big Picture - Functions and relations

The contents of this page are extremely foundational to the course. We will be weaving these foundations through almost every lesson in the course after this one.

2.1. Functions

Definition: A *function* is any method for taking a list of inputs and determining the corresponding output.

2.1.1. Examples of functions

Functions in mathematics: We can write functions with the usual notation from an algebra or calculus course:

- $f(x) = x^2 - 5$
- $g(x, y, z) = \frac{x^2 - y^2}{z}$

How is this a method for turning inputs into outputs? Given an input like $x = 2$, a function like f can find an output through the usual mechanism of substitution, more commonly called "plugging it in." Just substitute 2 into $f(x) = x^2 - 5$ to get $f(2) = 2^2 - 5 = -1$.

Functions in English: We can write functions in plain English (or any other natural language, but we'll use English). To do so, we write a *noun phrase,* and include blanks where the inputs belong:

- the capitol of ____
- the difference in ages between ____ and ____

How is this a method for turning inputs into outputs? Given an input like France, I can substitute it into "the capitol of ____" to get "the capitol of France" and use my knowledge to get Paris. If it were a capitol I didn't know, I could use the Internet to find out.

Functions in Python: We can write functions in Python (or other programming languages, but this course focuses on Python), like this:

```
def square ( x ):
    return x**2

def is_a_long_word ( word ):
    return len(word) > 8
```

How is this a method for turning inputs into outputs? I can ask Python to do it for me!

```
square(50)
```

```
2500
```

```
is_a_long_word( 'Hello' )
```

```
False
```

Functions in tables: Any two-column table can work as a function, if we follow a few conventions.

1. The left column will list the possible inputs to the function.
2. The right column will list the corresponding outputs.
3. Each input must show up only once in the table, so there's no ambiguity about what its corresponding output is.

Contents

Here's an example, which converts Bentley email IDs to real names for a few members of the Mathematical Sciences Department:

Bentley Email ID	Real Name
aaltidor	Alina Altidor
mbhaduri	Moinak Bhaduri
wbuckley	Winston Buckley
ncarter	Nathan Carter
lcherveny	Luke Cherveny

(We could add more names, but it's just an example.)

How is this a method for turning inputs into outputs? We use the familiar and fundamental operation of *lookup,* something that shows up in numerous places when working with data. (We'll return to the concept of lookup at the end of this chapter.) Given a Bentley Email ID as input, we look for it in the first column of the table, and once it's found, the appropriate output is right next to it in the right column.

Other types of functions: Later in the course we will see other ways to represent functions, but the ones above are the most common.

2.1.2. Which way is best?

The examples above show that you can express functions using math, English, Python, tables, and more. Although none of these ways is better than the others 100% of the time, we will typically give functions names and refer to them by those names. Examples:

- In Math: Rather than writing out $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ all the time, people just use the short name "the quadratic formula."
- In Python: The `def` keyword in Python is for giving names to functions so that you can use them later by just typing their name.

2.1.3. Why care about functions?

The concept of a function was invented because it represents an important component of how humans think about the processing of information. As you've seen above, functions show up in ordinary language, in mathematics, in tables of data, and code that processes data. Even people who don't do data work use functions unknowingly all the time when they talk about information, as in:

- I don't know all the state capitols. (In other words, I haven't memorized the function that gives the capitol for a state.)
- You better learn your times tables. (In other words, you should memorize the function that gives the product of two small whole numbers.)
- What's Kayla's phone number? (In other words, please apply the phone-number-of-person function to Kayla for me.)

Unsurprisingly, functions show up all over the place in data science. In particular, when working with a pandas DataFrame, we use functions often to summarize columns (such as compute the max, min, or mean) or to compute new columns, as in this example using Python's built in division function (written with the `/` symbol):

```
df['Per capita cost'] = df['Cost'] / df['Population']
```

2.2. Writing functions in Python

In MA346, we'll almost always want the functions we use to be written in Python, so that we can run them on data. Let's practice writing some functions in Python.

Exercise 1 - from mathematics

Write a function `solve_quadratic` that takes as input three real numbers a, b, and c, and gives as output a list of all real number solutions to the equation $ax^2 + bx + c = 0$. (It can return an empty list if there are no real number solutions.)

Example: `solve_quadratic(1,0,-4)` would yield `[-2,2]` because $1x^2 + 0x + (-4) = 0$ is the same equation as $x^2 = 4$.

The above exercise requires only the basic arithmetic built into Python, but when we do more advanced mathematics and statistics, we will import tools like numpy and scipy.

Exercise 2 - from English

Write a function `last_closing_price` that takes as input a NYSE ticker symbol and gives as output the price of one share at the last closing time of the NYSE. Hints:

- The URL https://finance.yahoo.com/quote/GOOG gives data for Alphabet, Inc. A similar URL works for any ticker symbol.
- You can extract all tables from a web page as pandas DataFrames as follows:

```
data_frames = pd.read_html( 'put the URL here' )
```

- That page has only one table, so it will be `data_frames[0]`.

Example: `last_closing_price('GOOG')` yielded something like `1465.85` in mid-June 2020.

It is not always guaranteed that you can turn an idea expressed in English, like "look up the last closing price of a stock," into Python code. For instance, no one knows how to write code that answers the question, "given a digital photo as input, return the year in which the photo was taken." But coming up with creative ways to answer important questions in code is a very valuable skill we will work to develop.

Exercise 3 - from a table

Write a function `country_capitol` that takes as input a string containing a country name and gives as output a string containing the name of the country's capitol. Hints:

- A list of countries and capitols appears here: https://www.boldtuesday.com/pages/alphabetical-list-of-all-countries-and-capitals-shown-on-list-of-countries-poster
- To convert two columns of a pandas DataFrame into a Python `dict` for easy lookup, try the following.

```
input_col = df['input column name']
output_col = df['output column name']
dictionary = dict( zip( input_col, output_col ) )
```

- You can then look items up using `dictionary[item_to_look_up]`, as in `dictionary['ZIMBABWE']`.

Example: `country_capitol('JORDAN')` would yield `'AMMAN'`.

Why do you think the `dict(zip())` trick given above works? What exactly is it doing?

2.3. Terminology

The following terminology is used throughout computing when discussing functions.

Definition: A *data type* is a category of values.

For instance, `int` is a Python data type for integers (that is, positive and negative whole numbers). Each number is a value in that data type. Other Python data types include `bool` (with the values `True` and `False`), `str` (short for "string" and containing text), and more.

Definition: A function's *input type* is the type of values you can pass as inputs when calling the function. If a function has multiple inputs, we might speak of its *input types* instead.

In Python, we are not required to write the input types of functions into our code, so we can only know them by reading a function's documentation or by inspecting the function's code and reasoning it out.

For example, the `square` function defined above probably has input type `float` (any number). The `is_a_long_word` function has input type `str`.

Definition: A function's *output type* is the type of values the function returns as outputs. Not all functions have a single return type, but many do.

For example, the `square` function always produces a `float` output and the `is_a_long_word` function alwayds produces a `bool` output.

These ides of input type and output type are a bit related to the ideas of domain and range of functions in mathematics, but they are not precisely the same. The difference is not important here.

Definition: A function is sometimes called a *map* from its input type to its output type. We say that a function *maps* its inputs to its outputs.

For instance, the `is_a_long_word` function maps strings to booleans.

Definition: A function that takes a single input is called a *unary* function. If it takes two inputs, it is a *binary* function. If it takes three inputs it is a *ternary* function. The number of inputs is called the *arity* of the function.

Although there are related words that go beyond three inputs (quaternary!) almost nobody uses them; instead, we would probably just say "a four-parameter function."

2.4. Relations

Definition: A *relation* is a function whose output type is `bool`, that is, the outputs are always either true or false.

2.4.1. Examples of relations

Relations in mathematics: Any equation or inequality in mathematics is a relation, such as $x^2 + y^2 \geq z^2$ or $x \geq 0$.

Consider $x \geq 0$. Given any input of the appropriate type, say $x = 15$, we can determine a true or false value by substitution. In this case, substituting $x = 15$ into $x \geq 0$ gives $15 \geq 0$, which we know is true. We could do a similar thing with $x^2 + y^2 \geq z^2$ if given three numerical inputs instead of just one.

Relations in English: Any declarative sentence with blanks in it is a relation. Here are two examples:

- ____ is the capitol of ____
- ____ is a fruit.

Given any input, you can use it to fill in the blank (or blanks) in the sentence and then judge (using your ordinary knowledge of the world and English) whether the sentence is true. For instance, if we're working with the sentence "____ is a fruit" and I provide the input "Python," then I get the sentence "Python is a fruit," which is obviously false, because it's a programming language, not a fruit.

Relations in Python: Any Python function with output type `bool` is a relation.

You can evaluate such relations by running them in Python, just as we did with functions earlier. In fact, the `is_a_long_word` function from earlier is not only a function, but also a relation. Here are two other examples:

```
def R ( a, b ):
    return a in b[1:]

def is_a_primary_color ( c ):
    return c in ['red','green','blue']
```

Although the first relation is an example with no clear purpose, the second one has a clear meaning. We can test it out like so:

```
is_a_primary_color( 'blue' ), is_a_primary_color( 'orange' )
```

```
(True, False)
```

Relations as lists: A very common way of defining a relation is to just list all the inputs for which the relation is true, and then we know that everything else makes it false.

In data science, we often do this using tables. For example, consider the table on the webpage mentioned in Exercise 3, above. That table lists all the pairs of inputs that make the "____ is the capitol of ____" relation true.

If you want to check whether, for example, "Bangalore is the capitol of India" is true, you can look to see if any row of the table is `('India','Bangalore')`. Since there is no such row, the relation is false for that input. (The capitol is actually New Delhi.)

Big Picture - Every table represents a relation.

Every table is a relation. Each row represents a set of inputs that would make the relation true, and any inputs that don't appear as a row in the table make it false.

Thus every pandas DataFrame is a relation, every SQL table is a relation, and every table you see printed in a book or on a webpage is a relation. This is why SQL is the language for querying *relational* databases.

The above big picture concept is almost 100% true. Technically, a pandas DataFrame or a SQL table can have repeated rows, which is unnecessary if you're defining a relation. And technically pandas DataFrames and SQL tables also have an extra layer of data called the "index" which we're ignoring for now, just concentrating on the contents of the table's columns.

Other types of relations: Later in the class we'll see even other ways to represent relations.

2.4.2. Which way is best?

Although we can express relations in all the ways just mentioned—in math, English, Python, or with lists—we typically *talk about* relations by using simple phrases. For instance, it's awkward to say "the '____ is a fruit' relation," so I would probably instead say something like "being a fruit." And instead of $x < y$, I might say something like "the usual less-than relation for numbers."

Sometimes we just use the central phrase to describe a binary relation. So to discuss the "____ has more employees than ____" relation, I might just use the phrase "has more employees than" when talking about it, or perhaps just "more employees." Usually it's clear what we mean.

2.4.3. Why care about relations?

The mathematical concept of a relation was invented because humans use it all the time when we think and speak, even though we don't precisely define it in everyday life. Every time we say a declarative sentence, this idea comes up. Here are some examples:

- If I say, "George isn't friends with Mia," then I'm relying on your familiarity with the being-friends-with relation, which you've known since Kindergarten.
- If I say, "Dell acquired EMC in 2015," then I'm relying on your familiarity with the "acquired" relation among companies, which you might not have been very familiar with before coming to Bentley.

The above examples are from binary relations, which are possibly the most common type. Just as a function can be binary (that is, take two inputs), so can a relation, because it's just a special type of function. But of course we can have unary relations as well (taking one input only), like the `is_a_long_word` and `is_a_primary_color` examples above, and we can have relations with three or more inputs as well.

A very important use of relations in data science is for *filtering* a dataset. We often want to focus our attention on just the section of a dataset we're interested in, which we describe as "filtering" to keep the rows we want (or "filtering out" the rows we don't want). In pandas, you can select a subset of a DataFrame `df` and return it as a new DataFrame (or, rather, a view on the original), like so:

```
# To filter the rows of a DataFrame, index the DataFrame with the relation:
df[put_any_relation_here]

# Here's an example, which uses the >= relation to filter for adults:
df[df['age'] >= 18]
```

2.5. Relations and functions in data

Exercise 4 - food inspections

The table below shows a sample of data taken from [a larger dataset on data.world about Chicago city food inspections](). Imagine the entire dataset of over 150,000 rows based on the sample of the first 10 rows shown below.

1. Name at least two relations expressed by the contents of this table. (You need not use all the columns.)
2. What are the input types, output type, and arity of each of your relations?
3. Does the table contain any sets of columns that define a function?
4. If so, what are the input types, output type, and arity of the function(s)?

Business	Address	Inspection Date	Inspection Type	Results
ZAM ZAM MIDDLE EASTERN GRILL	3461 N CLARK ST	11/07/2017	Complaint	Pass
SPINZER RESTAURANT	2331 W DEVON AVE	11/07/2017	Complaint Re-Inspection	Pass
THAI THANK YOU RICE & NOODLES	3248 N LINCOLN AVE	11/07/2017	License Re-Inspection	Pass
SOUTH OF THE BORDER	1416 W MORSE AVE	11/07/2017	License	Pass
BEAVERS COFFEE & DONUTS	131 N CLINTON ST	11/07/2017	License	Not Ready
BEAVERS COFFEE & DONUTS	131 N CLINTON ST	11/07/2017	License	Not Ready
FAT CAT	4840 N BROADWAY	11/07/2017	Complaint Re-Inspection	Pass
SAFARI SOMALI CUISINE	6319 N RIDGE AVE	11/07/2017	License	Fail
DATA RESTAURANT	2306 W DEVON AVE	11/06/2017	Complaint	Out of Business

Exercise 5 - tech companies

The table below shows a sample of data taken from a larger dataset on data.world about the 2016 Technology Fast 500. Imagine the entire dataset of 500 rows based on the sample of the first 10 rows shown below.

1. Name at least two relations expressed by the contents of this table. (You need not use all the columns.)
2. What are the input types, output type, and arity of each of your relations?
3. Does the table contain any sets of columns that define a function?
4. If so, what are the input types, output type, and arity of the function(s)?

CEO Name	City	Company Name	Country	Market	State
Charles Deguire	Boisbriand	Kinova Inc.	Canada	Canada	QC
Greg Malpass	Burnaby	Traction on Demand	Canada	Canada	BC
Jack Newton	Burnaby	Clio	Canada	Canada	BC
Jory Lamb	Calgary	VistaVu Solutions Inc.	Canada	Canada	AB
Wayne Sim	Calgary	Enersight	Canada	Canada	AB
Bryan de Lottinville	Calgary	Benevity, Inc.	Canada	Canada	AB
J. Paul Haynes	Cambridge	eSentire	Canada	Canada	ON
Jason Flick	Kanata	You.i TV	Canada	Canada	ON
Matthew Rendall	Kitchener	Clearpath	Canada	Canada	ON
Dan Latendre	Kitchener	Igloo Software	Canada	Canada	ON

2.6. Some technical notes

2.6.1. Connections between functions and relations

As you've probably noticed, there are some close relationships between relations and functions. Let's state them explicitly.

- Our definitions say that a relation is *a special kind of function;* that is, it's one whose output type has to be bool. So every relation is really also a function.
- But in the last two exercises, we've been thinking about relations and functions in tables. There we saw that we can think of a function as *a special kind of relation;* that is, it's one in which one column has all unique values, so that it can be used for input lookup in an unambiguous way.

2.6.2. Applying functions and relations

This idea of "input lookup" is called *applying* a function. For example, we apply the `country_capitol` function by looking up the country in the table and giving the corresponding capitol as output.

But we can actually do lookup in a relation as well, as long as we don't mind the possibility of getting more than one output. For instance, if we use the Technology Fast 500 table shown above and look up a city name, and ask for the corresponding company name, we won't always get just one answer. Even in just the small sample of the data we have, we can see that Calgary houses at least three different companies.

In short, functions let you apply them and get a unique answer, while relations let you apply them and get any number of answers.

2.6.3. Inverses

As mentioned above, a function is a relation in which *for each* input, *there is exactly one* output. But for *some* functions, the reverse is also true: For each *output*, there is exactly one *input*.

For example, consider the Technology Fast 500 table again, and let's assume that each company and CEO name is unique (i.e., there are not two CEOs name Jack Newton, or two companies named Clearpath, etc.). Consider the function that maps a company name to the corresponding CEO name; let's call it `lookup_ceo_for_company`.

- As with every function, for each input company, there is exactly one CEO output.
- But in this case, also, for each CEO output, there is exactly one input company.

While we chose to use the company as input and provide the CEO name as output, we could also have done it in the other order. That is, we could have created a function `lookup_company_for_ceo` that takes a CEO name as input and provides the corresponding company name as output. It just depends on which column you choose to use as the input and which you choose to use as the output.

This concept is probably familiar from mathematics, where we speak of *inverting* a function. In mathematical notation, we write the inverse of f as f^{-1}, but in computing, we can use more descriptive names, like the example of `lookup_ceo_for_company` and `lookup_company_for_ceo`.

In summary: For a relation to be a function, it has to provide just one output for each input. For it to be invertible, it has to have just one input for each output.

2.7. An extremely common data operation: Lookup

When working with data and writing code, we "look up" values in many different ways. We've already discussed above how applying a function expressed in a table is done by looking up the input and finding the corresponding output.

Let's review the most common ways that lookup operations show up in Python coding. Almost all of them use square brackets, because that's the common coding notation for looking up an item in a larger structure.

1. If we have a Python list `L` then we can look up the fourth item in it using the syntax `L[3]`, for example. In this way, you can think of a list as a function from numbers to the contents of the list.
2. If we have a Python dictionary `D` then we can look up an item in it using the syntax `D[my_item]`. So a dictionary is very much like a function; it maps its keys to their corresponding values.
3. If we have a pandas DataFrame, there are many ways to look up items in it, including:
    - filtering for just some rows, as discussed earlier, using syntax like `df[df.X==Y]`, and then selecting the column to use as the result, as in `df[df.Name=='Smith'].Employer`
    - choosing one or more rows and/or columns by their names, using `df.loc[rows,cols]`, as in `df.loc['May':'June','Rainfall']`
    - choosing one or more rows and/or columns by their zero-based index, using `df.iloc[rows,cols]`, as in `df.iloc[:,5]`

Some of the lookup operations shown above act like functions and some act like relations. For instance, a Python list always returns one value when you use square brackets for lookup, so that behaves like a function. But a pandas DataFrame might yield multiple values when you execute code like `df[df.Name=='Smith'].Employer`, because there may be many Smiths in the dataset. If you don't care about getting *all* the results, but want to just choose one of them, you can always add `.iloc[0]` on the end of the code to select just the first result from the list, as in `df[df.Name=='Smith'].Employer.iloc[0]`.

Later in the course we will see that SQL joins (called by various names in pandas, including merge, concat, and join) are highly related to all the lookup concepts just discussed. A SQL or pandas join is like doing many lookups all at once, which is why it is such a common operation.

By Nathan Carter