

# YData: Introduction to Data Science



Class 02: Introduction to Python

# Overview

Very quick review of the history of data science

Quick Discussion of the reading from “Everybody Lies”

Observational and Experimental data

## Intro to Python

- Expressions
- Names
- Call expressions (functions)
- Data types
  - Numbers and strings
- If there is time
  - Lists
  - Dictionaries



(a very incomplete list)

# Data



Ishango bone  
(20,000 BCE)



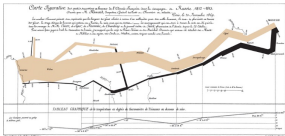
## Cuneiform tablets (4,000 BCE)



## Quipus in South America (1100-1500)

[illegible]

## Demographics (1600's)



## Golden age of data visualization (1850-1900)



## Big data (now)

# Probability

## Key Take Away

Probability models  
dominated data analysis  
prior to using  
computational methods

Initial development  
(1600's)

# Probability in Statistics

## (1820's – 1950's)

## Math Stats dominates (1900-1960's)

## "Big data"

# Computers

Abacus  
(2400 BCE)

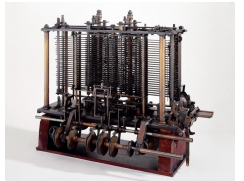


## Antikythera mechanism

(100 BCE)



## Analytical Engine (1800's)



## Hollerith Tabulating Machine (1890)

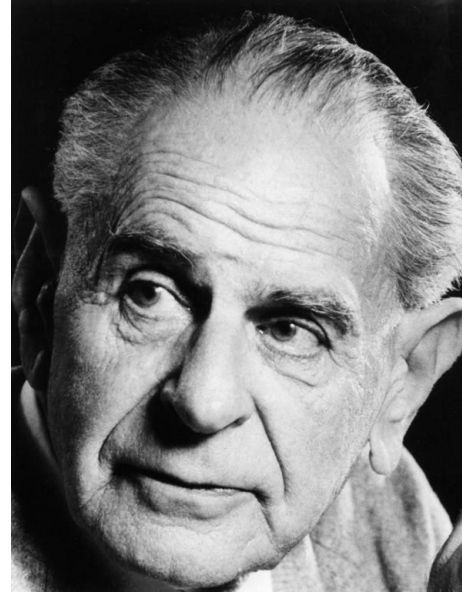
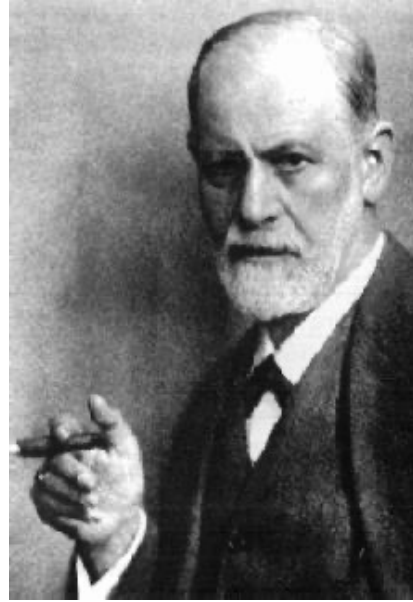


Mainframes, PCs, Internet,  
etc.

(1950-present)



# Thoughts on the reading from Everybody Lies?



Let's take ~3 minutes to discuss the paper in groups of 3-4 people

# Observational and Experimental Studies

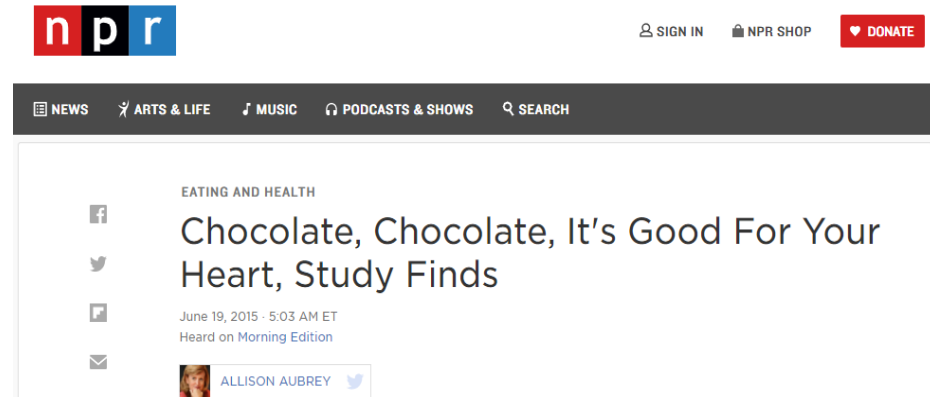
# Association vs. causation

**An association** is the presence of a reliable relationship between the treatments and outcome

- E.g., people who eat chocolate have lower rates of heart disease

**A causal relationship** is when changing the value of a treatment variable influences the value outcome variable

- E.g., consuming chocolate ***leads to*** a reduction in heart disease



There's a growing body of evidence suggesting that compounds found in cocoa beans, called polyphenols, may help protect against heart disease.  
Philippe Huguen/AFP/Getty Images



# THE SCIENCE NEWS CYCLE

JORGE CHAM © 2009

Start Here

## Your Research

Conclusion: A is correlated with B ( $p=0.56$ ), given C, assuming D and under E conditions.



...is translated by...

## UNIVERSITY PR OFFICE (YES, YOU HAVE ONE)



FOR IMMEDIATE RELEASE:  
SCIENTISTS FIND  
POTENTIAL LINK  
BETWEEN A AND B  
(UNDER CERTAIN CONDITIONS).

...which is then  
picked up by...

## NEWS WIRE ORGANIZATIONS



A CAUSES B, SAY  
SCIENTISTS.

...who are  
read by ...

## THE INTERNETS



[Scientists out to kill us again](#)

POSTED BY RANDOM GUY

Comments (377)

OMG! I kneeeew it!!

WTH???????

...then noticed by...

We saw it on a Blog!

A causes B all the time  
What will this mean for Obama?

BREAKING NEWS BREAKING NEWS BREA

## CNC Cable NEWS



...and caught  
on ...

## 4 LOCAL EYEWITLESS NEWS

WHAT YOU DON'T  
KNOW ABOUT "A"  
CAN KILL YOU!  
MORE AT 11...



...eventually  
making it to...

YOUR GRANDMA



Association  
does not  $\neq$   
causation!

# Confounding

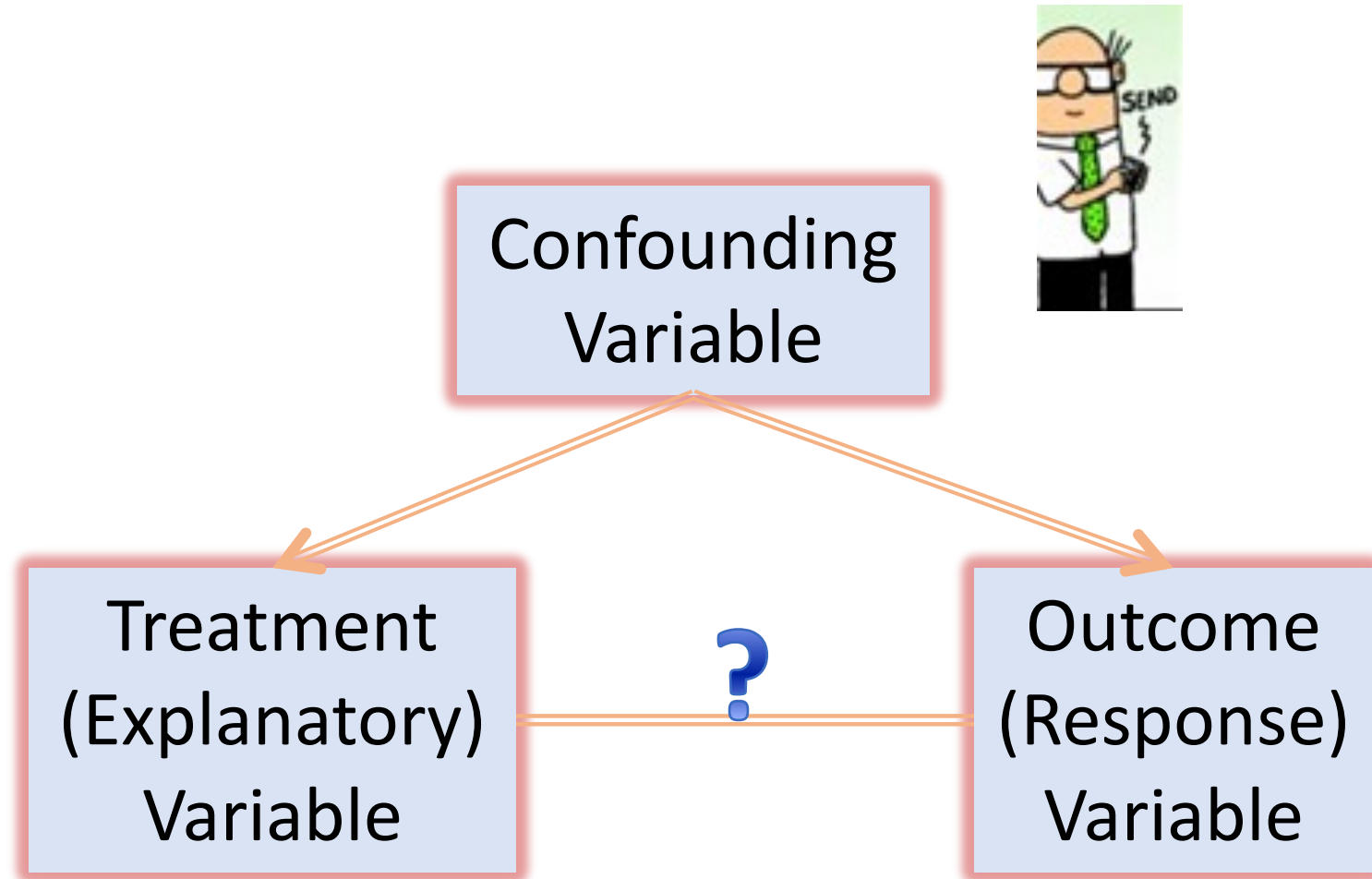
A **confounding variable** (also known as a **lurking variable**) is a third variable that is associated with both the treatment (explanatory) variable and the outcome (response) variable

A confounding variable can offer a plausible explanation for an association between the other two variables of interest





# Confounding



# Observational and experimental studies

An **observational study** is a study in which the researcher does not actively control the value of any treatment variable but simply observes the values as they naturally exist

An **experiment** is a study in which the researcher actively controls one or more of the treatment variables

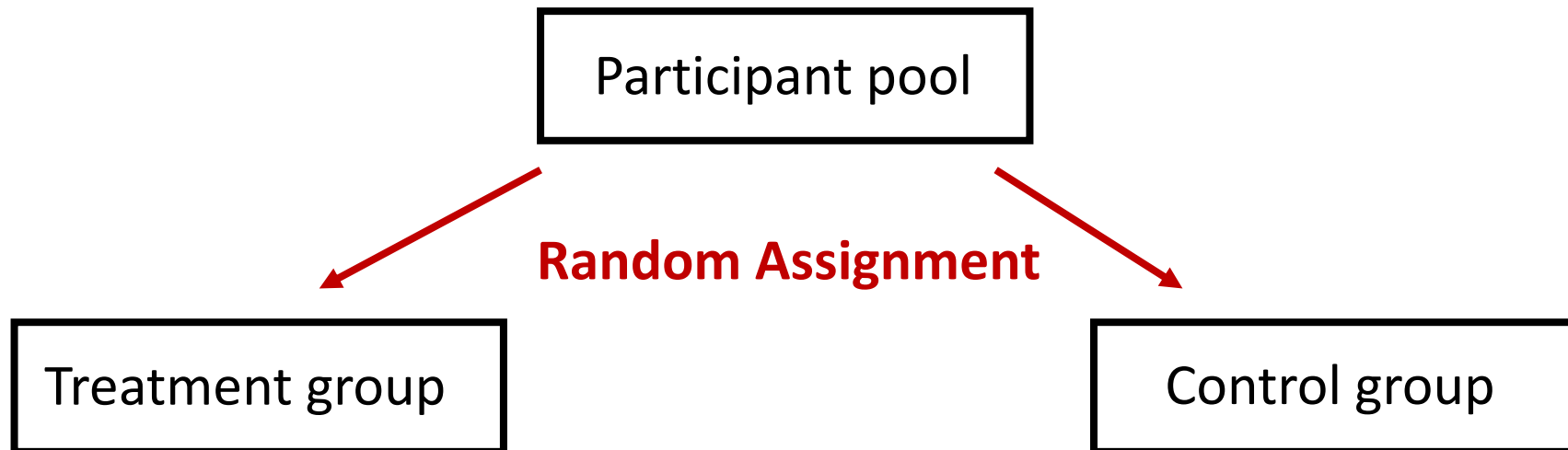
- Randomly assigns treatments to cases
- Allows one to get at questions of **causation**!



# Randomized Controlled Experiment

Take a group of participant and ***randomly assign***:

- Half to a *treatment group* where they get chocolate
- Half in a *control group* where they get a fake chocolate (placebo)
- See if there is more improvement in the treatment group compared to the control group



# Observational and experimental studies

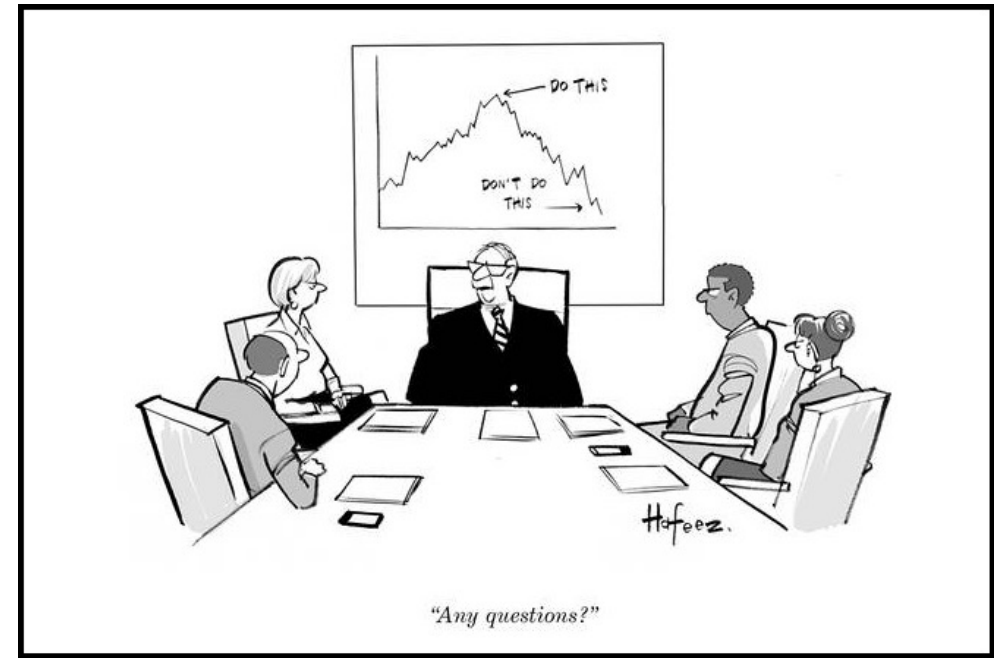
Experimental science (and data) grew in the 20<sup>th</sup> century, and dominates the sciences

Other fields is not possible or ethical to run experiments

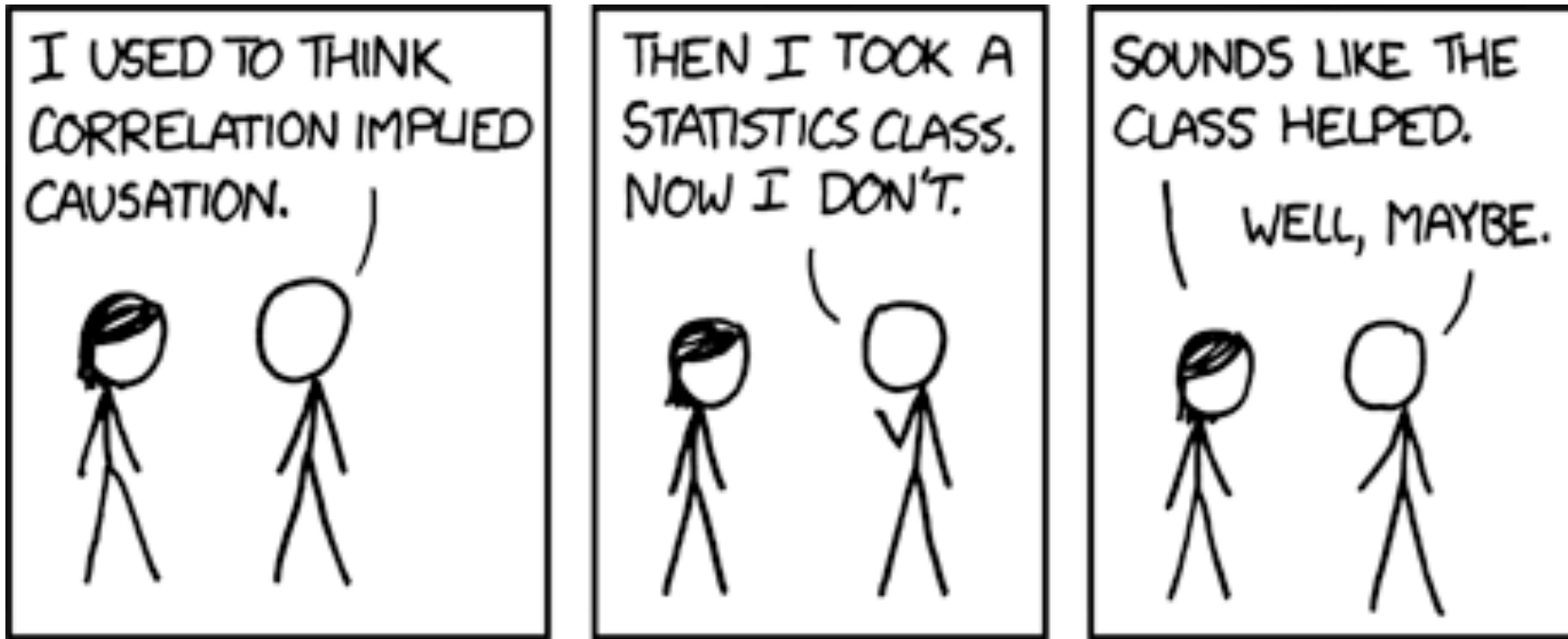
- E.g., Economics
  - although there are "natural" experiments

Data Science analyses often focus on observational data

- Be aware of the limitations in drawing causal conclusions from data!



# Questions?





# Programming in Python

Understanding the language fundamentals is important

Learn through practice, not only by reading or watching but by doing

- Like learning to ride a bike

Follow along with the class 2 Jupyter notebook!



# Let's start up Python!

Did everyone get Anaconda and the *ydata123\_2023e* environment installed?

If you weren't able to, you can follow the material in today's class using:

- Google colabs
- Mybinder



# Downloading the class code

If you have the *ydata123\_2023e* environment working, you can get the class code using:

- `import YData`
- `YData.download.download_class_code(2)`

If you are using colabs please run:

```
!pip install polars
```

```
!pip install https://github.com/emeyers/YData\_package/tarball/master
```

```
from google.colab import drive
```

```
drive.mount('/content/drive')
```

# Expressions

# Expressions

*Expressions* describe how a computer should combine pieces of data

- They are evaluated to by the computer and return a value
- E.g., mathematical expressions
  - Multiplication:  $3 * 4$
  - Exponentiation:  $3^{**}4$

| Operation      | Operation | Example     | Value |
|----------------|-----------|-------------|-------|
| Addition       | +         | $2 + 3$     | 5     |
| Subtraction    | -         | $2 - 3$     | -1    |
| Multiplication | *         | $2 * 3$     | 6     |
| Division       | /         | $7 / 3$     | 2.667 |
| Remainder      | %         | $7 \% 3$    | 1     |
| Exponentiation | **        | $2^{**}.05$ | 1.414 |



# Syntax

The *Syntax* of a language is its set of grammar rules for how expressions can be written

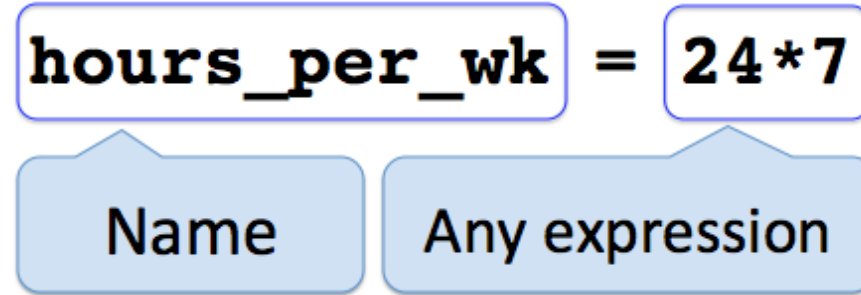
- *SyntaxError* indicates that an expression structure doesn't match any of the rules of the language.
- E.g., failed attempt at exponentiation:  $3 * * 4$

```
File "<ipython-input-2-012ea60b41dd>", line 1
  3 * * 4
    ^
SyntaxError: invalid syntax
```

Let's explore this in Jupyter!

Names

# Assignment statements



*Names* store the values (from an expression)

- i.e., they are like variables in algebra

Names are assigned values using the = symbol

- E.g., `my_number = 7`

Let's explore this in Jupyter!

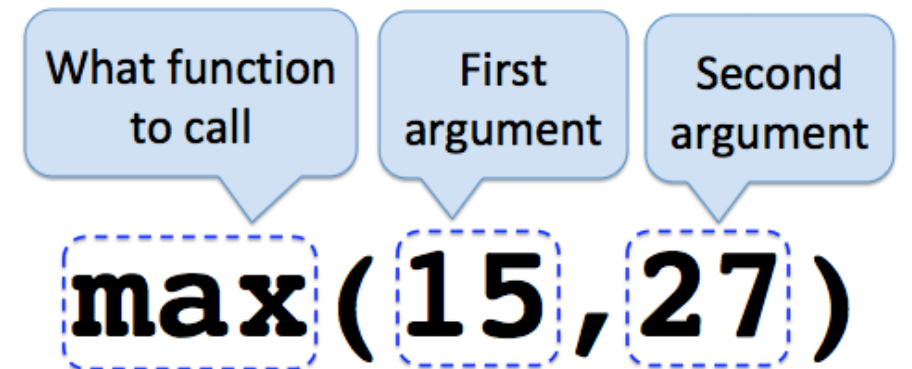
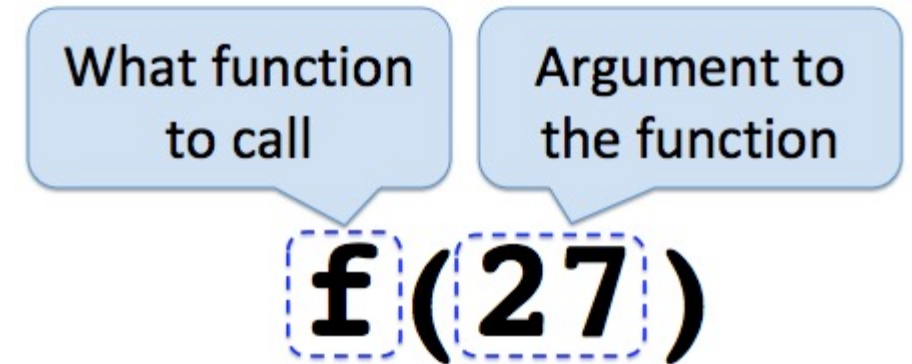
# Call Expressions

# Anatomy of a Call Expression

*Call expressions* are expressions that call functions

- Functions take in one or more values (arguments) and (usually) return another value

Example: taking the maximum value



Let's explore this in Jupyter!



Data types

# Arithmetic operations

| Operation      | Operation | Example    | Value |
|----------------|-----------|------------|-------|
| Addition       | +         | $2 + 3$    | 5     |
| Subtraction    | -         | $2 - 3$    | -1    |
| Multiplication | *         | $2 * 3$    | 6     |
| Division       | /         | $7 / 3$    | 2.667 |
| Remainder      | %         | $7 \% 3$   | 1     |
| Exponentiation | **        | $2 ** .05$ | 1.414 |

We can store the output of evaluating expression in names

- `my_result = 10 * 2`

# Numbers in Python: Ints and Floats

Python has two real number types

- **int**: an integer of any size
- **float**: a number with an optional decimal part

An int never has a decimal point - a float always does

- 3      *# int of float?*
- 2.7    *# int of float?*

A float might be printed using scientific notation

# Notes on Floats



## Three limitations of float values:

- They have limited size (but the limit is huge)
- They have limited precision of 15 - 16 decimal places
- After arithmetic, the final few decimal places can be wrong

Let's explore this in Jupyter!

# Arithmetic Question

What numbers are these expressions equal to?

A.  $3 * 10 ** 10$

B.  $10 * 3 ** 10$

C.  $(10 * 3) ** 10$

D.  $10 / 3 / 10$

E.  $10 / (3 / 10)$

A. 30000000000

B. 590490

C. 590490000000000

D. 0.333333333333333333337

E. 33.3333333333333333336

Best to err on the side of using parentheses!



# Strings

# Text and Strings

A string value is a snippet of text of any length

- 'a'
- 'word'
- "there can be 2 sentences. Here's the second!"

Strings consisting of numbers can be converted to numbers

- int('12')
- float('1.2')

Any value can be converted to a string

- str(5)

Let's explore this in Jupyter!

# Discussion Questions

Assume you have run the following statements

- `x = 3`
- `y = '4'`
- `z = '5.6'`

What's the source of the error in each example?

- A. `x + y`
- B. `x + int(y + z)`
- C. `str(x) + int(y)`
- D. `str(x, y) + z`

Types

# Every value has a type

We've seen several types so far:

- int: `2`
- Built-in function: `abs()`
- float: `2.2`
- str: `'Red fish, blue fish'`

The type function can tell you the type of a value

- `type(2)`
- `type('Red fish')`

An expression's type is based on its value, not how it looks

- `x = 2`
- `type(x)`

Let's explore this in Jupyter!

# Conversions

Strings that contain numbers can be converted to numbers

- `int('12')`
- `float('1.2')`
- `float('one point two')`      `# Not a good idea!`

Any numeric value can be converted to a string

- `str(5)`

Numbers can be converted to other numeric types

- `float(1)`
- `int(1.2)`      `# DANGER: loses information!`

# Lists

# Lists

Lists are ways to store multiple items

We can create lists using square brackets []

- `my_list = [2, 3, 4]`

We can also access list items using square brackets []

- `my_list[2]`

Lists can contain elements of different types

- `my_list2 = [5, 6, 'seven']`

Let's explore this in Jupyter!

**TO DO LIST**  
**1. make lists**  
**2. look at lists**  
**3. PANIC!**



# Dictionaries

# Dictionaries



Dictionaries allow you to look up ***values*** based on a ***key***

- i.e., you supply a “key” and the dictionary returns the stored value

We can create dictionaries using the syntax:

- `my_dict = { 'key1': 5, 'key2': 20 }`

We can retrieve dictionary values by supplying a key using square brackets []

- `my_dict['key2']`

Let's explore this in Jupyter!