Statements

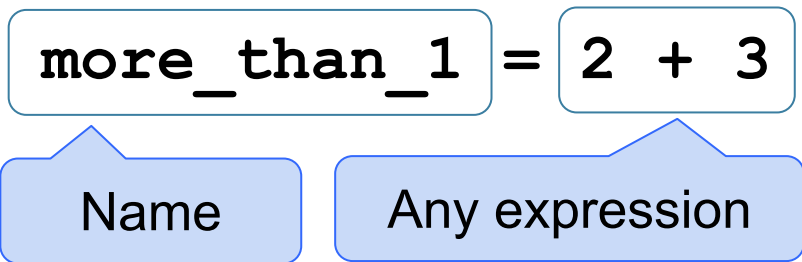$$\boxed{\texttt{more\_than\_1}} = \boxed{\texttt{2 + 3}}$$
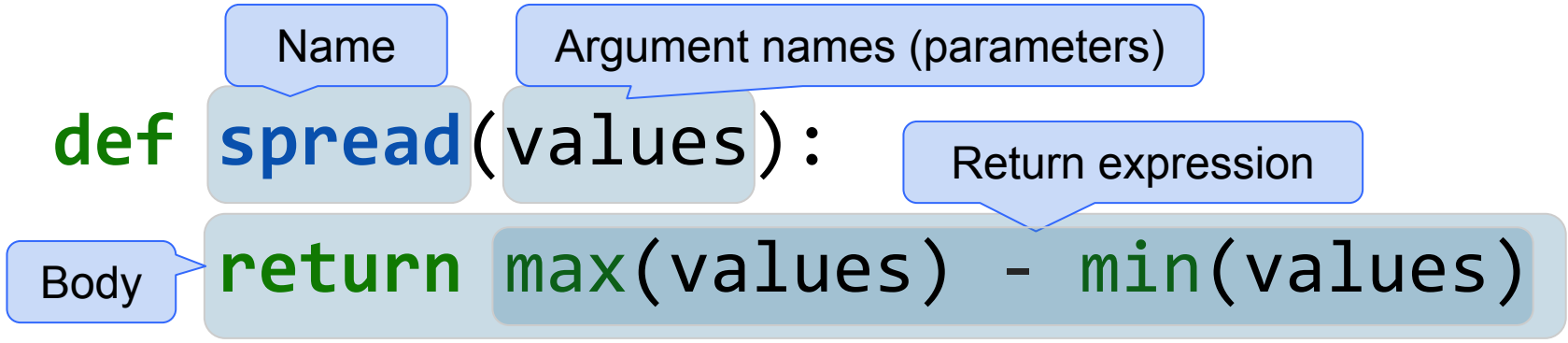
Name / Any expression

- Statements don't have a value; they perform an action
- An assignment statement changes the meaning of the name to the left of the = symbol
- The name is bound to a value (not an equation)

- < and > mean what you expect (less than, greater than)
- <= means "less than or equal"; likewise for >=
- == means "equal"; != means "not equal"
- Comparing strings compares their alphabetical order

Arrays - sequences that can be manipulated easily
- All elements of an array should have the same type
- Arithmetic is applied to each element of an array individually
- Elementwise operations can be done on arrays of the same size

Name / Argument names (parameters) / Return expression / Body

```
def spread(values):
    return max(values) - min(values)
```

For Statements

```
for  i  in      np.arange(12)        :
    print(i)
```

The body is executed **for** every item in a sequence
The body of the statement can have multiple lines
The body should do something: print, assign, hist, etc.

Conditional Statements

```
if <if expression>:
    <if body>
elif <elif expression 0>:
    <elif body 0>
elif <elif expression 1>:
    <elif body 1>
...
else:
    <else body>
```

Growth Rate: the rate of increase per unit time
- After one time unit, a quantity **x** growing at rate **g** will be
$$\texttt{x * (1 + g)}$$
- After **t** time units, a quantity **x** growing at rate **g** will be
$$\texttt{x * (1 + g) ** t}$$
- If **after** and **before** are measurements of the same quantity taken **t** time units apart, then the growth rate is
$$\texttt{(after/before) ** (1/t) - 1}$$

Values in Tables: Every column of a table is an array.
- **Categorical**
  - May or may not have an ordering
  - Categories are the same or different
  - Allows grouping by value (**group, pivot, join**)
- **Numerical**
  - Ordered
  - Allows binning by value (**bin, hist**)

Binning is counting the number of numerical values that lie within ranges, called bins.
- Bins include the lower bound and exclude the upper bound
- Values equal to the upper bound of a bin go into the next bin
- The upper bound of a bin is the lower bound of the next bin

163, 168, 170, 171, 173, 183, 185, 188, 189, ...
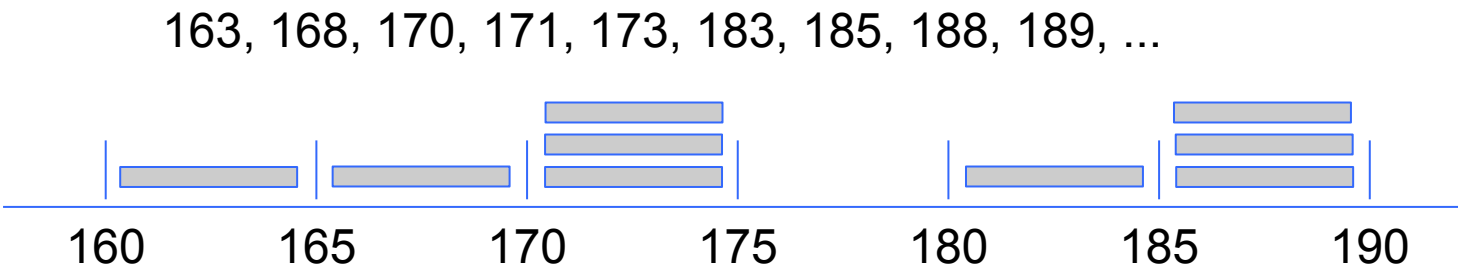
A **histogram** has two defining properties:
- The bins are contiguous (though some might be empty) and are drawn to scale
- The **area** of each bar is equal to the proportion of entries in the bin

Has total area 1 (or 100%)
Vertical axis units: Proportion / Unit on the horizontal axis

---

- A histogram of proportions of all possible outcomes of a *known* random process is called a *probability histogram*
- A histogram is a summary visualization of a *distribution*
- A histogram of proportions of actual outcomes generated by sampling or actual data is called an *empirical histogram*
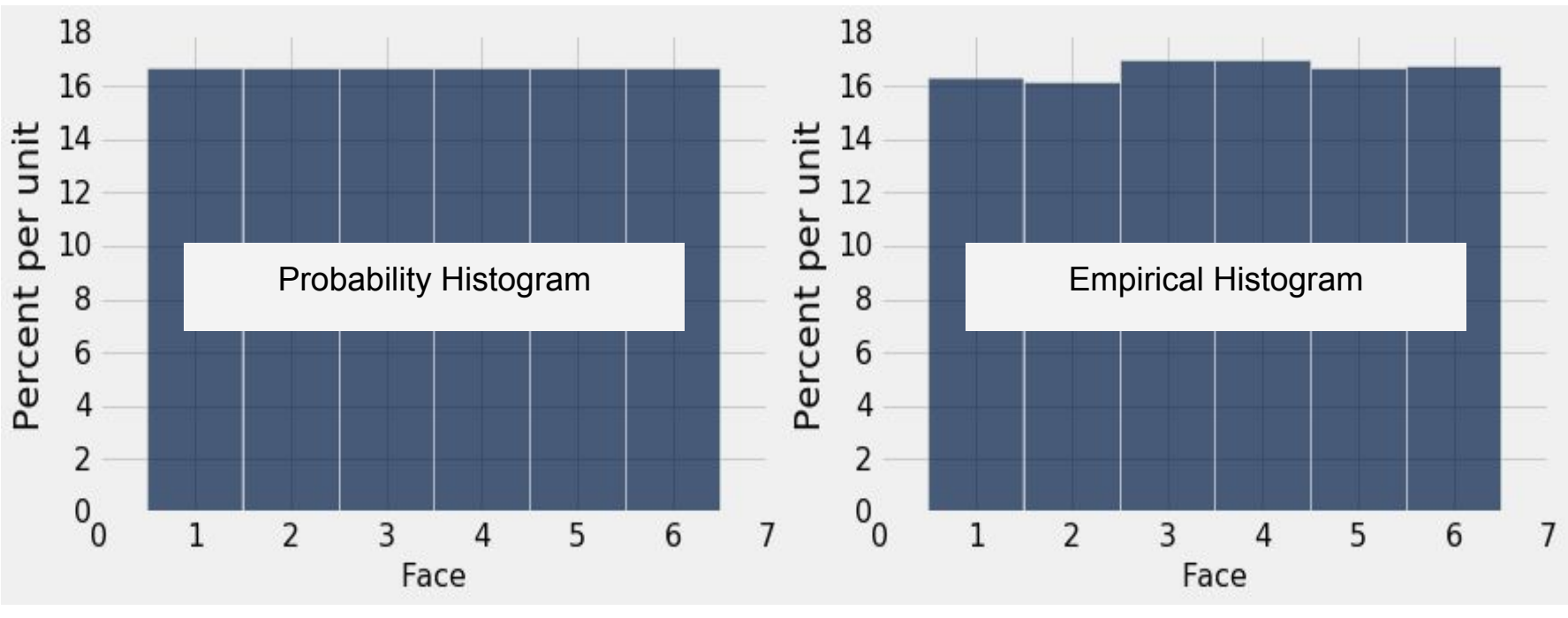
Calculating Probabilities

Complement Rule: P(event does not happen) = 1 - P(event happens)

Multiplication Rule: P(two events happen) = P(one happens) * P(other happens, given the first happened)

Addition Rule: P(an event happens) = P(first way it can happen) + P(second way it can happen) IF it can happen in ONLY one of two ways

In the examples in the left column, np refers to the NumPy module, as usual. Everything else is a function, a method, an example of an argument to a function or method, or an example of an object we might call the method on. For example, `tbl` refers to a table, `array` refers to an array, and `num` refers to a number. `array.item(0)` is an example call for the method `item`, and in that example, `array` is the name previously given to some array.

Example function call	Value of a call to the function
`max(array); min(array)`	Maximum or minimum of a list or array
`sum(array)`	Sum of all elements in a list or array; The sum of boolean values is the number that are True
`len(array)`	Length (num elements) in a list or array
`round(num); np.round(array)`	The nearest integer to a single number or each number in an array
`abs(num); np.abs(array)`	The absolute value of a single number or each number in an array
`np.average(array), np.mean(array)`	The average of the values in an array
`np.arange(start, stop, step)` `np.arange(start, stop)` `np.arange(stop)`	An array of numbers starting with `start`, going up in increments of `step`, and going up to but excluding `stop`. When `start` and/or `step` are left out, default values are used in their place. Default `step` is 1; default `start` is 0.
`array.item(index)`	The item in the array at some index. `array.item(0)` is the first item of `array`.
`np.append(array, item)`	A copy of the array with item appended to the end. If item is another array, all of its elements are appended.
`np.random.choice(array)`	An item selected at random from an array.
`np.random.choice(array, n)`	An array of n items selected at random with replacement from an array.
`Table()`	An empty table.
`Table.read_table(filename)`	A table with data from a file.
`tbl.num_rows`	The number of rows in a table.
`tbl.num_columns`	The number of columns in a table.
`tbl.labels`	A list of the column labels of a table.
`tbl.with_column(name, values)` `tbl.with_columns(n1, v1, n2, v2…)`	A table with an additional or replaced column or columns. `name` is a string for the name of a column, `values` is a list or array.
`tbl.column(column_name_or_index)`	The values of a column (an array).
`tbl.select(col1, col2, …)`	A table with only the selected columns. (Each argument is the label of a column, or a column index.)
`tbl.drop(col1, col2, …)`	A table without the dropped columns. (Each argument is the label of a column, or a column index.)
`tbl.relabeled(old_label, new_label)`	A new table with a label changed.
`tbl.take(row_indices)`	A table with only the rows at the given indices. `row_indices` is an array of indices.
`tbl.sort(column_name_or_index)`	A table of rows sorted according to the values in a column (specified by name/index). Default order is ascending. For descending order, use argument `descending=True`. For unique values, use `distinct=True`.
`tbl.where(column, predicate)`	A table of the rows for which the column satisfies some predicate. See "Table.where predicates" below.
`tbl.apply(function, column)`	An array where a function is applied to each item in a column.
`tbl.group(column_or_columns)`	A table with the counts of rows groupued by unique values or combinations of values in a column or columns.
`tbl.group(column_or_columns, func)`	A table that groups rows by unique values or combinations of values in a column or columns. The other values are aggregated by `func`.
`tblA.join(colA, tblB, colB)` `tblA.join(colA, tblB)`	Generate a table with the columns of `tblA` and `tblB`, containing rows for all values of a column that appear in both tables. Default `colB` is `colA`. `colA` is a string specifying a column name, as is `colB`.
`tbl.pivot(col1, col2, vals, collect)` `tbl.pivot(col1, col2)`	A pivot table where each unique value in `col1` has its own column and each unique value in `col2` has its own row. The cells of the grid contain row counts (two arguments) or the values from a third column, aggregated by the collect function.
`tbl.sample(n)` `tbl.sample(n, with_replacement)`	A new table where n rows are randomly sampled from the original table. Default is with replacement. For sampling without replacement, use argument `with_replacement=False`. For non-uniform sample, provide `weights=distribution` where `distribution` is an array containing the probability of each row.
`tbl.scatter(x_column, y_column)`	Draws a scatter plot consisting of one point for each row of the table.
`tbl.barh(categories)` `tbl.barh(categories, values)`	Displays a bar chart with bars for each category in a column, with height proportional to the corresponding frequency. `values` argument unnecessary if table has only a column of categories and a column of values.
`tbl.hist(column, units, bins)`	Generates a histogram of the numerical values in a column. `units` and `bins` are optional arguments, used to label the axes and group the values into intervals (bins), respectively. Bins have the form [a, b).

Operations: addition 2+3=5; subtraction 4-2=2; division 9/2=4.5 multiplication 2*3=6; division remainder 11%3=2; exponent 2**3=8

Data Types: string 'hello'; **boolean** True, False; **int** 1, -5; **float** - 2.3, -52.52, 7.9

Arithmetic with arrays is elementwise:
`make_array(1,2,3) ** 2` *# [1, 4, 9]*

Table.where predicates (x is a string or number)
`are.equal_to(x)`
`are.not_equal_to(x)`
`are.above(x)` *# val > x*
`are.below(x)` *# val < x*
`are.between(x, y)` *# x <= val < y*
`are.contained_in([x, y, z])` *# val is either x, y, or z*

Exam Scratch Paper

Exam Scratch Paper