Data 6 Python Cheat Sheet

This cheat sheet has been modified from the Data 6 Python Reference and includes all of the functions and table methods that you will need for the final exam.

Built-In Python Functions

Function	Description	Input	Output
`str(val)`	Converts `val` to a string	A value of any type	The value as a **string**
`int(num)`	Converts `num` to an int	A numerical value (**string** or **float**)	The value as an **int**
`float(num)`	Converts `num` to a float	A numerical value (**string** or **int**)	The value as a **float**
`len(arr)`	Returns the length of `arr`	**array** or **list**	**int**: the length of the array or list
`max(arr)` or `max(arr)`	Returns the maximum or minimum value in `arr`	**array** or **list**	The maximum or minimum value the array (usually an **int**)
`sum(arr)`	Returns the sum of the values in `arr`	**array** or **list**	**int** or **float**: the sum of the values in the array
`abs(num)`	Returns the absolute value of `num`	**int** or **float**	**int** or **float**

NumPy Array Functions

Function	Description	Input	Output
`make_array(val1, val2, ...)`	Makes a NumPy array with the inputted values	A sequence of values	An **array** with those values
`np.mean(arr)` or `np.average(arr)`	Calculates the average value of `arr`	An **array** of numbers	**float**: array average

`np.sum(arr)`	Returns the sum of the values in `arr`	**array**	**int** or **float**: sum of array values
`np.arange(stop)`, `np.arange(start, stop)`, or `np.arange(start, stop, step)`	Creates an array of numbers starting at `start`, going up in increments of `step` (default is 1), and going up to but excluding `stop`.	**int** or **float**	**array**

Tables and Table Methods

Function	Description	Input	Output
`tbl.with_column(name, values)`	Adds an extra column onto `tbl` with the label `name` and `values` as the column values	1. **string**: name of the new column 2. **array**: values in the column	**Table**: a copy of the original table with the new column
`tbl.column(col)`	Returns the values in a column	**string** or **int**: the column name or index	**array**: the values in that column
`tbl.num_rows`, `tbl.num_columns`	Computes the number of rows or columnsin `tbl`	None	**int**: number of rows or columns in the table
`tbl.select(col1, col2, ...)`	Creates a copy of `tbl` only with the selected columns	**string** or **int**: the column name(s) or index(es) to be included in the table	**Table** with the selected columns
`tbl.drop(col1, col2, ...)`	Creates a copy of `tbl` without the selected columns	**string** or **int**: the column name(s) or index(es) to be dropped from the table	**Table** without the selected columns
`tbl.sort(column_name)`	Sorts the rows of `tbl` by the values in the `column_name` column.	1. **string** or **int**: name or index of the column to sort	**Table**: a copy of the original table sorted by the column

		2. (Optional) `descending=True`	
`tbl.where(column, predicate)`	Creates a copy of `tbl` containing only the rows where the value of `column` matches the `predicate`. See `Table.where` predicates below.	1. **string** or **int**: column name or index 2. `are.(...)` predicate	**Table**: a copy of the original table with only the rows that match the predicate
`tbl.take(row_indices)`	Creates a table with only the rows at the given indices. `row_indices` is either an array of indices or an integer corresponding to one index.	**int** or **array**: indices of rows to be included in the table	**Table**: copy of the original table with the rows at the given indices
`tbl.apply(function)` or `tbl.apply(function, col1, col2, ...)`	Returns an **array** of values resulting from applying a function to each item in a column.	1. **Function**: function to apply to column 2. (Optional) **string** or **int**: the column name(s) or index(es) to apply the function to	**array** containing an element for each value in the original column after applying the function
`tbl.group(column, function)`	Groups rows in `tbl` by unique values in a column. Values in the other columns are aggregated by count (by default) or the optional argument `function`.	1. **string**: column on which to group 2. (Optional) **Function**: function to aggregate values in cells (defaults to counting rows)	**Table** a new groupped table
`tbl.pivot(col1, col2, values, collect)`	Creates a pivot table where each unique value in `col1` has its own column and each unique value in `col2` has its own row. Counts or aggregates values from a third column, collected with some function.	1. **string**: column in `tbl` for the pivot table columns 2. **string**: column in `tbl` for the pivot table rows 3. (Optional) **string**: column in `tbl` for the pivot table values	**Table**: a new pivot table

		4. (Optional) **Function**: how the values are collected	
`tblA.join(colA, tblB, colB)`	Generate a table with the columns of `tblA` and `tblB`, containing rows for all values in `colA` and `colB` that appear in `tblA` and `tblB`, respectively.	1. **string**: name of column in `tblA` 2. **Table**: the other table 3. (Optional) **string**: the name of the shared column in `tblB`	**Table**: a new combined table

Visualization Functions

Function	Description	Input	Output
`tbl.barh(categories)` or `tbl.barh(categories, values)`	Displays a horizontal bar chart with bars for each category in the column `categories`. `values` specifies the column corresponding to the size of each bar.	1. **string**: name of the column with categories 2. (Optional) **string**: name of categories column	None: draws a bar chart
`tbl.hist(column)`	Generates a histogram of the numerical values in `column`.	**string**: name of the column	None: draws a histogram
`tbl.plot(x_column, y_column)` or `tbl.plot(x_column)`	Draws a line plot consisting of one point for each row in `tbl`. If only `x_column` is specified, `plot` will plot the rest of the columns on the y-axis with different colored lines.	1. **string**: name of the column on the x-axis 2. **string**: name of the column on the y-axis	None: draws a line graph
`tbl.scatter(x_column, y_column)`	Draws a scatter plot consisting of one point for each row in `tbl`.	1. **string**: x-axis column 2. **string**: y-axis column	None: draws a scatter plot

Table.where Predicates

These functions can be passed in as the second argument to `tbl.where(..)` and act as a condition by which to select rows from `tbl`.

Predicate	Description
`are.equal_to(Z)`	Equal to `Z` (can be an **int**, **float** or **string**)
`are.above(x)`, `are.above_or_equal_to(x)`	Greater than (or equal to) `x`
`are.below(x)`, `are.below_or_equal_to(x)`	Less than (or equal to) `x`
`are.between(x,y)`, `are.between_or_equal_to(x,y)`	Greater than (or equal to) `x`, and less than (or equal to) `y`
`are.strictly_between(x,y)`	Greater than `x` and less than `y`
`are.contained_in(A)`	True if it is a substring of `A (if` `A` `is a **string**)` `or an element of` `A` `(if` A` is an **array**)
`are.containing(S)`	Contains the string `S`

Conditional Statements and Iteration

Syntax	Description

`if <if expression>:`
  `<if body>`
`elif <elif expression>:`
  `<elif body>`
`else:`
  `<else body>`
	Executes the code in `<if body>` only if `<if expression>` evaluates to `True`. If `<if expression>` is `False`, checks `<elif expression>` and executes code in `<elif body>` if `True`. Otherwise, executes the code in `<else body>`

`for <element> in <sequence>:`
  `<for body>`
	Repeats code in `<for body>` for each `<element>` in `<sequence>` (array, string, etc.), assigning `<element>` to each value in `<sequence>` one at a time

`while <boolean expression>:`
  `<while_body>`
	Repeats code in `<while body>` while `<boolean expression>` is `True`