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 exams.

Built-In Python Functions

Function	Description	Input	Output
`str(val)`	Converts `val` to a string	A value of any type (`int`, `float`, `NoneType`, etc.)	The value as a string
`int(num)`	Converts `num` to an integer	A numerical value (represented as a `string` or `float`)	The value as an `int`
`float(num)`	Converts `num` to a float	A numerical value (represented as a `string` or `int`)	The value as an `float`
`len(arr)`	Returns the length of `arr`	`array` or `list`	`int`: the length of the array or list
`max(arr)`	Returns the maximum value in `arr`	`array` or `list`	The maximum value of the array
`min(arr)`	Returns the minimum value in `arr`	`array` or `list`	The minimum value of the array
`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`
`print(input, ...)`	Prints the input values, separated by spaces	`input`: any inputs to print	`None`
`type(object)`	Returns the type of `object`	`object`: the object whose type is to be determined	`type`: the type of the object

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`: The average of the array
`np.sum(arr)`	Returns the sum of the values in `arr`	**array**	`int` or `float`: the sum of the values in the array
`np.prod(arr)`	Returns the product of the values in `arr`	**array**	`int` or `float`: the product of the values in the array
`np.sqrt(num)`	Calculates the square root of `num`	`int` or `float`	`float` : the square root of the number

`np.arange(stop)`, `np.arange(start, stop)`, or `np.arange(start, stop, step)`	Creates an array of sequential numbers starting at `start` , going up in increments of `step` , and going up to but excluding `stop`. Default `start` is 0, default `step` is 1	`int` or `float`	`array`
`np.count_nonzero (arr)`	Returns the number of non-zero (or `True`) elements in an array	An `array` of values	`int`: the number of non-zero values in `arr`
`np.append(arr, item)`	Appends `item` to the end of `arr`. Does not modify the original `arr`	1. `array` to append 2. Item to append (any type)	`array`: a new array with the appended item
`np.cumsum(arr)`	Returns the cumulative sum of the elements in `arr` , where each element is the sum of all preceding elements including itself	`array`	`array`: the cumulative sum of the values in the array
`np.diff(arr)`	Computes differences between consecutive elements in `arr`	`array`	`array`: the differences between consecutive elements in the array containing `len(arr) - 1` elements
`np.sort(arr)`	Sorts an array in ascending order.	`array`	`None` (`arr` is sorted; no copy is made)

Tables and Table Methods

Function	Description	Input	Output
`Table()`	Creates an empty table, usually to extend with data	None	**Table**: An empty Table
`Table().read_table(filename)`	Create a table from a data file	`string`: the name of the file	**Table**: a table created from the data file
`tbl.with_column(name, values)` or `tbl.with_columns(n1, v1, n2, v2, ...)`	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(s)
`tbl.column(col)`	Returns the values in a column in `tbl`	`string` or `int`: the column name or index	**array**: the values in that column
`tbl.num_rows`	Compute the number of rows in `tbl`	None	**int**: the number of rows in the table
`tbl.num_columns`	Compute the number of columns in `tbl`	None	**int**: the number of columns in the table
`tbl.labels`	Returns the labels in `tbl`	None	**array**: the names of each column as strings
`tbl.select(col1, col2, ...)`	Creates a copy of `tbl` only with the selected columns	`string` or `int`: column name(s) or index(es)	**Table** with the selected columns
`tbl.drop(col1, col2, ...)`	Creates a copy of `tbl` without the selected columns	`string` or `int`: column name(s) or index(es)	**Table** without the selected columns
`tbl.relabeled(old_label, new_label)`	Creates a new table, changing the column name specified by `old_label` to `new_label`, and leaves the original table unchanged	1. `string`: old column name 2. `string`: new column name	**Table**: a copy of the original table with the changed column name
`tbl.show(n)`	Displays the first `n` rows of `tbl`. Defaults to showing the entire table	(Optional) `int`: number of rows to display	**None** (table is displayed)
`tbl.sort(column_name[, descending=True])`	Sorts rows by `column_name`. Ascending by default unless `descending=True` is included	1. `string` or `int`: name or index of the column to sort 2. (Optional) `descending=True`	**Table**: a copy of the original table with the column sorted
`tbl.where(column, predicate)`	Creates a copy of `tbl` containing only the rows where the value of `column` matches the `predicate`	1. `string` or `int`: column 2. (Optional) `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	`int` or `array`: indices of rows to be included in the table	**Table**: a copy of the original table with only the rows at the given indices

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` (`Z` can be an `int`, `float`, or `string`).
`are.not_equal_to(Z)`	Not equal to `Z` (`Z` can be an `int`, `float`, or `string`).
`are.above(x)`	Greater than `x`.
`are.above_or_equal_to(x)`	Greater than or equal to `x`.
`are.below(x)`	Less than `x`.
`are.below_or_equal_to(x)`	Less than or equal to `x`.
`are.between(x, y)`	Greater than or equal to `x` and less than `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`.