

36-350 Statistical Computing R/Python Cheatsheet

stdout	<code>print("R code!")</code>	<code>print("Python code!")</code>
find type	<code>typeof(x)</code>	<code>type(x)</code>
removing vars	<code>rm(x)</code> <code>exists('x')</code>	<code>del x</code>
vectors		
instantiation	<code>x <- c(0, 0, 0, 0, 0)</code> <code>x <- rep(0, 5)</code> <code>x <- vector("integer", 5)</code> <code>x <- integer(5)</code> <code>x <- seq(1, 5, by = 1)</code> <code>x <- seq(1, 5, length.out = 5)</code> <code>x <- 1:5</code>	<code>x = np.zeros(5, dtype = int)</code> <code>x = np.ones(5, dtype = float)</code> <code>x = np.full(5, 5.43)</code> <code>x = np.arange(0, 10, 2)</code> <code>x = np.linspace(1, 5, 5)</code> <code>x = np.array(["a", "b", "c", "d", "e"])</code>
usage	<code>length(x)</code> <code>x[1]</code> # 1-indexed <code>x[1:2]</code> # first 2 <code>x[-y]</code> # x without indices y <code>rev(x)</code> <code>sort(x)</code> <code>order(x)</code> <code>as.vector(x)</code> # cleaning attrs <code>sum(x, na.rm=TRUE)</code> <code>unique(x)</code> <code>table(x)</code> <code>union(x, y)</code> <code>intersect(x, y)</code> <code>setdiff(x, y)</code> <code>setequal(x, y)</code> <code>is.element(x, y)</code> # x[i] in y? <code>match(x, y)</code>	<code>x.size</code> <code>x[0]</code> # 0-indexed <code>x[b:e:s]</code> # [begin, end) with step, accepts negatives. WARNING: returns a "view," like a pointer instead of a copy <code>np.delete(x, y)</code> <code>np.sort(x); x.sort()</code> # sorted copy; in-place <code>np.argsort(x)</code> # indices that would sort an array <code>np.nansum(x)</code> <code>np.unique(x)</code> <code>np.unique(x, return_counts=True)</code> <code>np.union1d(x, y)</code> <code>np.intersect1d(x, y)</code> <code>np.setdiff1d(x, y)</code> <code>set(x) == set(v)</code> <code>np.in1d(u, v)</code>
rand	<code>set.seed(5)</code> <code>runif(8)</code>	<code>np.random.seed(5)</code> <code>np.random.random(8)</code>
logical subsetting	<code># let x: vector[int]</code> <code>x > 0: vector[bool]</code> <code>x[x>0 & x<0.4]</code> # elements of x where x>0 and x<0.4 <code>which(x < 0)</code>	<code># let x: np.array[int]</code> <code>x > 0: np.array[bool]</code> <code># can do like R with much parentheses, but consider np.logical_...</code> (https://stackoverflow.com/questions/33384529/difference-between-numpy-logical-and-and) <code>np.where(x < 0)</code>
edge case data types	<code>NA; is.na()</code> # missing data <code>NULL; is.null()</code> # fns that return nothing <code>NaN; is.nan()</code> # e.g., 0/0 <code>Inf; -Inf; is.infinite()</code> # e.g., 1/0 <code># NAN IS STRICT SUBSET OF NA</code>	<code># no NA in numpy</code> <code>None; is None</code> # fns that return nothing <code>np.nan; np.isnan()</code> # e.g., 0/0 <code>np.inf; np.isinf()</code> # e.g., 1/0
lists/dicts	<code>list(foo=1:5, bar=c("a", "b"))</code> <code>x[[2]]</code> # column 2: "a" "b" <code>x[["bar"]]</code> # equivalent <code>unlist(x)</code> # list elements -> a vector <code>data.frame(</code> <u>u = 1:2,</u> <u>v = c("a", "b")</u> <code>)</code> # dfs are square lists <code>matrix(1:6, nrow=2)</code> # matrices are dfs w/ cols of same type # filled col by col <code>x[2, 1]; x[2,]</code> # matrix indexing is standard	<code>{"foo": np.arange(1, 6), "bar": np.array(["a", "b"])}</code> <code># use pandas for this use case.</code>