36-350 Statistical Computing R/Python Cheatsheet

stdout

```r
print("R code!")
```

```python
print("Python code!")
```

find type

```r
typeof(x)
```

```python
type(x)
```

removing vars

```r
rm(x)
exists('x')
```

```python
del x
```

vectors

instantiation

```r
x <- c(0, 0, 0, 0, 0)
x <- rep(0, 5)
x <- vector("integer", 5)
x <- integer(5)
x <- seq(1, 5, by = 1)
x <- seq(1, 5, length.out = 5)
x <- 1:5
x <- c(a=1, b=2, c=3)
attr(x, "creator") <- "pef"
```

```python
x = np.zeros(5, dtype = int)
x = np.ones(5, dtype = float)
x = np.full(5, 5.43)
x = np.arange(0, 10, 2)
x = np.linspace(1, 5, 5)
x = np.array(["a", "b", "c", "d", "e"])
```

usage

```r
length(x)
x[1] # 1-indexed
x[1:2] # first 2
x[-y] # x without indices y
rev(x)
sort(x)
order(x)
as.vector(x) # cleaning attrs
sum(x, na.rm=TRUE)
unique(x)
table(x)
union(x, y)
intersect(x, y)
setdiff(x, y)
setequal(x, y)
is.element(x, y) # x[i] in y?
match(x, y)
```

```python
x.size
x[0] # 0-indexed
x[b:e:s] # [begin, end) with step, accepts negatives. WARNING: returns a "view," like a pointer instead of a copy
np.delete(x, y)

np.sort(x); x.sort() # sorted copy; in-place
np.argsort(x) # indices that would sort an array

np.nansum(x)
np.unique(x)
np.unique(x, return_counts=True)
np.union1d(x, y)
np.intersect1d(x, y)
np.setdiff1d(x, y)
set(x) == set(v)
np.in1d(u, v)
```

rand

```r
set.seed(5)
runif(8)
```

```python
np.random.seed(5)
np.random.random(8)
```

logical subsetting

```r
# let x: vector[int]
x > 0: vector[bool]
x[x>0 & x<0.4] # elements of x where x>0 and x<0.4
which(x < 0)
```

```python
# let x: np.array[int]
x > 0: np.array[bool]
# can do like R with much parentheses, but consider np.logical_... (https://stackoverflow.com/questions/33384529/difference-between-numpy-logical-and-and)
np.where(x < 0)
```

edge case data types

```r
NA; is.na() # missing data
NULL; is.null() # fns that return nothing
NaN; is.nan() # e.g., 0/0
Inf; -Inf; is.infinite() # e.g., 1/0
# NAN IS STRICT SUBSET OF NA
```

```python
# no NA in numpy
None; is None # fns that return nothing
np.nan; np.isnan() # e.g., 0/0
np.inf; np.isinf() # e.g., 1/0
```

lists/dicts

```r
list(foo=1:5, bar=c("a", "b"))
x[[2]] # column 2: "a" "b"
x[["bar"]] # equivalent
x$bar # sugar
unlist(x) # list elements -> a vector
data.frame(
  u = 1:2,
  v = c("a", "b")
) # dfs are square lists
matrix(1:6, nrow=2)
  # matrices are dfs w/ cols of same type
  # filled col by col
x[2, 1]; x[2, ] # matrix indexing is standard
```

```python
{"foo": np.arange(1, 6), "bar": np.array(["a", "b"])}
# can use pandas for this use case.
np.arange(1, 7).reshape([3, 2])
  # filled row by row
```

control flow

```r
seq_along(x)

break
next
```

```python
# it's python,  figure it out

break
continue
```

exception handling

```r
message(<str>)
warning(<str>)
stop(<str>)

tryCatch(
  {
    <code that may fail>
  },
  error = function(c) { <deal with error here> },
  warning = function(c) { <deal with warning here> },
  message = function(c) { <issue a message here> },
  finally = { <stuff to do regardless of whether there is an error> }
)
try(<code>) # bad idea :(
```

```python
raise Exception("generic")
raise TypeError("more specific")

try:
  print(x)
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")
```

unit tests

```r
library(testthat)

<stuff...>

test_that(desc = "Test for expected output length (1)",
          expect_length(char2int("a"), 1))
test_that(desc = "Test that output is proper type (integer)",
          expect_type(char2int("a"), "integer"))
test_that(desc = "Test for input string of length greater than 1",
          expect_error(char2int("aa")))
test_that(desc = "Test for improper input type (numeric)",
          expect_error(char2int(1)))
```

```python
import unittest

<stuff...>

class testchar2int(unittest.TestCase):
  def test_olen(self):
    self.assertEqual(char2int("a"), 1)
  def test_otype(self):
    self.assertIsInstance(char2int("a"), int)
  def test_oupper(self):
    self.assertEqual(char2int("A"), 1)
  def test_longstr(self):
    with self.assertRaises(Exception): char2int('aa')
  def test_numstr(self):
    with self.assertRaises(Exception): char2int('1')

if __name__ == '__main__':
  unittest.main()
```