Interpreted Interactive Scientific Programming Languages

Selective and Incomplete Reference Card

	Python	R	IDL	Matlab/Octave
Start session	`ipython -pylab`	`RStudio`	`idl`	`octave -q`
Run code from file	`execfile('file.py')` or `run file.py`	`source('file.R')`	`file.pro` or `.run file.pro`	`file`
Command history	`hist -n`	`history()`	`help,/rec`	`history`
Save command history		`savehistory(file=..)`	`journal,'file'`	`diary on` and `diary off`
End session	`Ctrl-D or sys.exit()`	`q()`	`exit or Ctrl-D`	`exit or quit`
Install packages	`pip install ..`	`install.package()`		
Show/Set working path	`sys. ..`	`getwd() / setwd(..)`	`cd,..,current=curr`	
Comment sign	`#`	`#`	`;`	

Help	Python	R	IDL	Matlab/Octave
Interactive	`help()`	`help.start()`	`?`	`doc`
Help on function f	`help(f)` or `?f` or `??f`	`help(f)` or `?f`	`?f`	`help f`
Demonstration examples		`demo()` or `example(plot)`	demo	
Search more help		`help.search('plot')` or `apropos('plot')`		`lookfor plot`
Locate function	`help(plot)`	`find(plot)`		`which plot`
List methods	`dir()`	`methods(plot)`		
Others		`str()`		
Data Summary		`describe()`, `summary()`		

Syntax	Python	R	IDL	Matlab/Octave
Assignment	`a=1; b=1`	`a<-1; b<-2`	`a=1 & b=2`	`a=1; b=2;`
Procedure				
Function				
Return value	`return x`	`x` (last statement)	`return, x`	as assigned
Object Method				
Operator Precedence	`func(); x[ind:ind]; x[ind]; x.attr; **; *,/,%; +,-; <,>,<=,>=,!=,==; in,not in; not,and,or`			
Terminal Output	`print(..)`	`print()`	`print, ..`	
Terminal Input				
In place: $a = a + b$	`a+=b`		`a+=b`	`a+=b`
In place ops.			`+= *= /= #= ##= &&= ..`	

Variables/Types	Python	R	IDL	Matlab/Octave
Basic types	`int, long, float, complex, bool, str, tuple, list, dict`	`character, numeric, integer, complex, logical`	`int, long, float, complex, string, double, byte, ..`	
Conversion	`int(), float()`	`as.list, ..`	`fix(), float(), ..`	
Type Checks		`is.na(x)`		
Not a number	`nan`	`NaN, NA`	`!values.f_nan`	`NaN`
Infinity	`inf, plus_inf`		`!values.f_infinity`	`Inf`
Complex number 2+i	`2+1j`	`2+1i`	`complex(2,1)`	`2+i`
Real/Imaginary	`np.real(z), z.real, z.imag`	`Re(z), Im(z)`	`real_part(z), imaginary(z)`	`real(z), imag(z)`
Abs/Argument	`abs(z)`	`abs(z), Arg(z)`	`abs(z), atan(z,/ph)`	`abs(z), arg(z)`
Complex conjugate	`np.conj(z)`	`Conj(z)`	`conj(z)`	`conj(z)`

Math	**Python**	**R**	**IDL**	**Matlab/Octave**
Basic	`+ - * /` or `//`	`+ - * /` or `%/%`	`+ - * /`	`+ - * /` or `.* ./`
Power a^b	`a**b`	`a^b`	`a^b`	`a.^b`
Modulo	`np.mod(a,b)`	`a%b`	`a mod b`	`rem(a,b)`
Factorial $a!$	`m.factorial(a)`	`factorial(a)`	`factorial(a)`	
Round	`np.round()`, `np.ceil()`, `np.floor()`	`round()`, `ceil()`, `floor()`	`round()`, `ceil()`, `floor()`	`round()`, `ceil()`, `floor()`
Round towards zero	`np.fix()`		`fix()`	`fix()`
Trigonometry	`sin()`, `cos()`, `tan()`, `asin()` or `arcsin()`, `acos()` or `arccos()`, `atan()` or `arctan()` or `atan2()`			
Hyperbolic	`sinh()`, `cosh()`, `tanh()`			
Others	`np.sqrt()`, `np.log()`, `np.exp()`	`sqrt()`, `log()`, `exp()`	`sqrt()`, `alog()`, `exp()`	`sqrt()`, `log()`, `exp()`
Constants	`m.pi`, `m.e`	`pi`, `exp(1)`	`!pi`, `exp(1)`	`pi`, `exp(1)`

Relational				
Basic	`== < > <= >= !=`	`== < > <= >= !=`	`eq lt gt le ge ne`	`== < > <= >= ~=`

<table>
<tr><th>Logical</th><th></th><th></th><th></th><th></th></tr>
<tr><td>Single-elem</td><td><code>and or</code></td><td><code>&& ||</code></td><td><code>&& ||</code></td><td><code>&& ||</code></td></tr>
<tr><td>Element-wise</td><td><code>and or</code></td><td><code>& |</code></td><td><code>and or</code></td><td><code>& |</code></td></tr>
<tr><td>XOR</td><td></td><td><code>xor(a,b)</code></td><td><code>a xor b</code></td><td><code>xor(a,b)</code></td></tr>
<tr><td>NOT</td><td><code>not a</code></td><td><code>!a</code></td><td><code>~a</code></td><td><code>~a !a</code></td></tr>
<tr><td>True if any nonzero</td><td></td><td></td><td></td><td><code>any(a)</code></td></tr>
<tr><td>True if all nonzero</td><td></td><td></td><td></td><td><code>all(a)</code></td></tr>
</table>

Vectors and Matrices	**Python**	**R**	**IDL**	**Matlab/Octave**
Vector 1,3,-4,10	`np.array([1,3,-4,10])`	`c(1,3,-4,10)`	`[1,3,-4,10]`	`[1; 3; -4; 10]`
Sequence 1,2,..,10	`range(1,11)`, `np.arange(1,11)`	`1:10`, `seq(1,10)`	`indgen(10)+1`	`1:10`
1,4,7,10	`np.arange(1,11,3)`	`seq(1,10,3)`	`indgen(4)*3+1`	`1:3:10`
Linearly spaced	`np.linspace(1,10,7)`	`seq(1,10,len=7)`		`linspace(1,10,7)`
Zeros $\begin{bmatrix}0&0&0&0\\0&0&0&0\end{bmatrix}$	`np.zeros((2,4))`	`matrix(0,2,4)`	`fltarr(4,2)`	`zeros(2,4)`
Ones $\begin{bmatrix}1&1&1&1\\1&1&1&1\end{bmatrix}$	`np.ones((2,4))`	`matrix(1,2,4)`	`fltarr(4,2)+1`	`ones(2,4)`
Identity matrix	`np.identity(3)`	`diag(3)`	`identity(3)`	`eye(3)`
Diagonal matrix	`np.diag([3,4,5])`	`diag(c(3,4,5))`	`diag_matrix([3,4,5])`	`diag([3 4 5])`
Reverse	`a[::-1]`	`rev(a)`	`reverse(a)`	`reverse(a)`
Transpose	`a.T`	`t(a)`	`transpose(a)`	`a.'`
Conjugate transpose	`a.T.conj()`	`Conj(t(a))`	`conj(transpose(a))`	`a'`
Flatten to 1D	`np.ravel(a)`		`a[*]`	
Flatten by rows	`a.flatten()`	`as.vector(t(a))`		`a'(:)`
Flatten by columns	`a.flatten(1)`	`as.vector(a)`	`(transpose(a))[*]`	`a(:)`
Flatten upper triang.		`a[row(a)<=col(a)]`		`vech(a)`
Reshape by rows	`a.reshape(2,-1)` or `a.setshape(2,3)`	`matrix(a,nrow=3, byrow=T)`	`reform(a,[2,3])`	reshape(a,3,2)'
Reshape by columns	`a.reshape(-1,2).T`	`matrix(a,nrow=2)`	`reform(transpose(a),[2,3])`	`reshape(a,2,3)`
Flip left-right (mirror horiz)	`np.fliplr(a)` or `a[:,::-1]`	`a[,4:1]`	`reverse(a)`	`fliplr(a)`
Flip up-down (mirror vert)	`np.flipud(a)` or `a[::-1,]`	`a[3:1,]`	`reverse(a,2)`	`flipud(a)`
Rotate 90 deg	`np.rot90(a)`		`rotate(a,1)`	`rot90(a)`
Repeat matrix	`np.kron(np.ones((2,3)),a)`	`kronecker(matrix(1,2,3),a)`		`repmat(a,2,3)`
Upper/lower triangular	`np.triu()`, `np.tril()`	`a[lower.tri(a)]`, `a[upper.tri(a)]`		`triu()`, `tril()`
Assign to all	`a.fill(3)`, `a[:]=3`		`a[*]=3`	`a(:)=3`
Concatenate	`np.concatenate((a,b))`	`c(a,b)`	`[a,b]`	`a b`
Repeat [1,2,1,2,1,2]	`np.concatenate((a,a,a))`	`rep(a,imes=3)`	`[a,a,a]`, `replicate(a,3)`	`[a a a]`
Repeat [1,1,1,2,2,2]	`np.repeat(a,3)`	`rep(a,each=3)`		
Max/Min	`np.max(a)`, `np.min(a)`	`max(a)`, `min(a)`	`max(a)`, `min(a)`	`max(a)`, `min(a)`
Max/Min position	`np.argmax(a)`	`which.max(a)`	`v = max(a,pos=i)`	`[v,i] = max(a)`

Continued on the next page...

Vectors and Matrices	Python	R	IDL	Matlab/Octave
Sum (over 1th dim)	np.sum(a,axis=0)	colSum(a)	total(a,1)	sum(a)
Sum (all elems)	np.sum(a)	sum(a)	total(a)	sum(sum(a))
Cumulative sum	np.cumsum(a)	cumsum(a)	total(a,/cum)	cumsum(a)
Dimensions	a.shape	dim(a)	size(a)	size(a)
Specific dimensions	a.ndim, a.nbytes, len(a)	ncol(a), object.size(a)	n_elements(a)	length(a), ndims(a)

Matrices	Python	R	IDL	Matlab/Octave
Input $\begin{bmatrix}2 & 3\\ 4 & 4\end{bmatrix}$	np.array([[2,3], [4,5]])	array(c(2,3,4,5), c(2,2))	[[2,3],[4,5]]	[2 3; 4 5]
Bind rows	np.vstack((a,b))	rbind(a,b)	[[a],[b]]	[a; b]
Bind columns	np.hstack((a,b))	cbind(a,b)	[a,b]	[a, b]
Bind slices(3d)	np.dstack((a,b))		[[[a]],[[b]]]	
Create single vector	np.concatenate(a,b, axis=None)		[a[*],b[*]]	[a(:),b(:)]
Elementwise operatoins	* / + -	* / + -	* / + -	.* ./
Matrix multiplication	matrixmultipy(a,b)	a%*%b	a#b a##b	a*b
Cross product				
Kronecker product	np.kron(a,b)	kronecker(a,b)		kron(a,b)
Solve linear equations	linalg.solve(a,b)	solve(a,b)	cramer(a,b)	a\b

Indexing	Python	R	IDL	Matlab/Octave
First row	a[0,]	a[1,]	a[*,0]	a(1,:)
Element 2,3 (row, col)	a[1,2]	a[2,3]	a[2,1]	a(2,3)
Array indices			a[[0,3],[0,2]]	a([1 3],[1 4])
Selection	a[2:]	a[-1]	a[1:*]	a(2:end)
Last element	a[-1]		a[n_elements(a)-1]	a(end)
Last 3 rows	a[-3:,:]		a[*,n_elements(a)-3:*]	a(end-2:end,:)
Every k-th row	a[::k,:]		a[ind,*]	a(1:2:end,:)
k-th in last dim	a[...,k]			
All except row 2, col 3		a[-2,-3]		
Diagonal elems	a.diagonal()		diag_matrix(a)	
Clipping	(a>90).choose(a,90) a.clip(min=2,max=90)	a[a>90]<-90	a<90 2>a<90	a(a>90)=90

Mathematical indexing: row-number × column-number: $M_{3\times4} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix}$

Sorting	Python	R	IDL	Matlab/Octave
Sorted values	np.sort(a)	sort(a)	a[sort(a)]	sort(a)
Sorted indices	np.argsort(a)	order(a)	sort(a)	

Linear Algebra	Python	R	IDL	Matlab/Octave
Element-wise multipl.	a*b	a*b	a*b	a.*b
Scalar product $\mathbf{a}\cdot\mathbf{b}$	np.dot(a,b)		transpose(a)#b	dot(a,b)
Cross product $\mathbf{a}\times\mathbf{b}$	np.dot(a,b)		crossp(a,b)	

Find	Python	R	IDL	Matlab/Octave
Nonzero indices	np.nonzero(a)[0] or np.where(a!=0)[0]	which(a!=0,arr.ind=T)	where(a ne 0)	[i j]=find(a)
Return values	a.compress((a>5).flat)	which(a>5)	a[where(a gt 5)]	find(a>5)

Random Numbers	Python	R	IDL	Matlab/Octave
Uniform	np.random.rand()	runif()	randomu()	rand()
Normal	np.random.randn()	rnorm()	randomn()	randn()

Python

1. stats: `scipy.stats.binned_statistic` - moving/rolling stats
2. mutable/non-mutable objects: assignment with copy (deepcopy)
3. Bitwise operations: `<<, >>, &, |, ~, ^`
4. Selected functions:
   - `bin(n), oct(n), hex(n)` - number to binary, octal, hexadecimal digits string
   - `ord(c), chr(n)` - integer code of character (unicode or ascii), and reversed
   - `int(s,n)` - base-n digits string to number (e.g. int('0011',2))
5. Numpy
   - Array left-most dim: row; second from left: column.
   - Concatenate at 1th (row-wise), 2nd (column-wise), 3rd dimensions: `np.vstack((a,b))`, `np.hstack((a,b))`, `np.dstack((a,b))` (since numpy 1.10): concatenate at n-th dimension: `np.stack((a,b),n)`
   - Tile (repeat axes): `np.tile(a, [1,4])`; Repeat individual elements (flattened): `np.repeat(a, 4)`
   - For multidimensional data, use np.array (not np.matrix); and then use .dot(), .conj(), and .T methods and attributes.
   - Slicing doesn't return a new array, but a view of of the original. That is, changing b=a[:,1] with b[0]=x will change a as well.
   - C-like index order: M[i,j], where j changes fastest → M[0,:] is printed first, as row, M[1,:] is printed next, etc.
   - Selecting a dimension in array will always reduce the rank, leading e.g. from 2-d to 1-d (or rank) matrix. Transpose doesn't do anything on 1-d array.
   - Array can treat rank-1 arrays as either row or column vectors. For example, `dot(A,v)` treats v as a column vector, while `dot(v,A)` treats v as a row vector.
     The usual mathematical column vector has shape (n,1). To generate, use e.g.: `c_[x]` or `x.reshape((n,1))` or `x[:,newaxis]`.
   - Fast array creation: `r_[:10]` or `r_[:10.]` or `r_[:10:2]`. Both, `r_[]` and `c_[]` actually createy arrays by stacking numbers along a row or a column, and *allow* using the ":" range slicing operator.
   - Submatrix indexing possible, e.g. with `ix_[(ind,ind)]` form.
   - Linear indexing is not straightforward. First, reshape to a linear sequence, then perform the operations, and then reshape to original size.
   - For array operations along one dimension, use the *axis=* keyword. It starts with 0: axis=0 for operating on each column, axis=1 for operating on each row.
   - *Indexing* keyword (e.g. meshgrid): 'ij' = matrix, 'xy' = cartesian (default). Ideally, use *indexing='ij'* explicitly:
     `A=meshgrid(:M,:N,:P) => A.shape==(N,M,P); meshgrid(:M,:N,:P,indexing='ij') => A.shape==(M,N,P)`
   - Attributes: *ndim, shape, size, dtype, itemsize, T.*
   - Creation functions: *array(), zeros(), ones(), eye(), arange(), linspace(), logspace(), empty(), random.random(), copy(), identity(), mgrid(), ogrid(), r()* (they usually take tuples for shape).
   - Shape-changing methods: *reshape(), ravel(), transpose(), T, resize().* If a -1 is given in a reshaping operation, the other dimensions are automatically calculated! Others: `atleast_2d(), mat(), newaxis`.
   - Concatenating: *vstack, hstack, column_stack, row_stack, concatenate.*
   - Questions: *all, any, nonzero, where.*
   - Ordering: *argmax, argmin, argsort, max, min, ptp, searchsorted, sort.*
   - Operations: *choose, compress, cumprod, cumsum, inner, fill, imag, real, prod, put, putmask, sum.*
   - Statistics: *cov, mean, std, var.*
   - Linear algebra: *dot, cross, outer, svd, vdot.*
   - Others: *histogram()* (similar to IDL)
   - More on fancy indexing and more tricks see/click: NumPy Tutorial or NumPy for Matlab Users.
6. Pylab / Matplotlib
   - `figsize(x,y) e.g. figsize(13,4)`
   - `subplot(yxn) e.g. subplot(121)`
7. My conventions:
   - Multi-dimensional data: [az, rg, pol-channels], if channels should be closer together (for block processing along azimuth).

R

1. loop functions: `map; apply, sapply, ..`
2. modules: `?modname, install.packages`, library(psych)
3. info on data set: `str(data), summary(data), head(data), psych.describe(data)`
4. accessing list elements: use double square brackets! e.g L[[1]] (using single brackets will give you back a list again)
5. smoothScatter(x,y) – great function!!!! 2-d histogram with densities!
6. plot(x,y,col=rgb(0,0,0,0.2),pch=19) – if many points, adding transparency helps to see where there more and where less points!

Additional Notes

1. Lexical scope: Python, R — Dynamic scope: IDL, Matlab.
2. Array/matrix indexing:
   - Zero-based: Python, IDL, C/C++
   - One-based: R, Matlab
   - Column-major format (data in memory column-by-colum), which is the standard mathematical matrix notation: Python, C/C++. This *C-like* index order changes the last axis index fastest.
     The fastest-changing axis/dimension is read/written/printed to screen first, therefore, a print command in column-major format will print first a[0,:] on first line, then a[1,:] on second line, etc. If characterizing indices with i,j as in a[i,j], then for the print, x=j, y=i.
     $$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \end{bmatrix} \tag{1}$$
   - Row-major format (row-by-row, useful in image processing): IDL, Fortran. This *Fortran-like* index order changes the first axis index fastest.
     $$\begin{bmatrix} a_{0,0} & a_{1,0} \\ a_{0,1} & a_{1,1} \end{bmatrix} \tag{2}$$

mxn, 2014.
Last compiled: April 17, 2016, 19:36.