Preamble

For each section, this cheatsheet presents a summary of Python functionalities. All informations presented here are extracted from python.org and scipy.org. For more details about each function, do not hesitate to read the full documentation online !

1 General Python

1.1 Built-in functions

abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

1.2 Python keywords

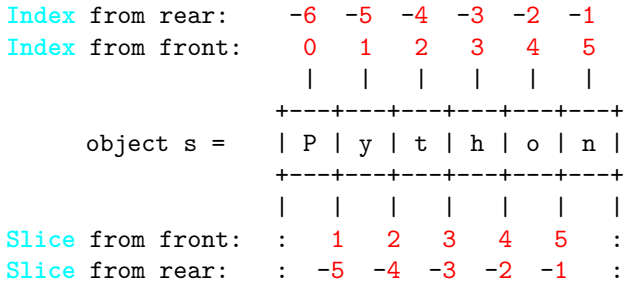
True	assert	del	for	in	or	while
False	break	elif	from	is	pass	with
None	class	else	global	lambda	raise	yield
and	continue	except	if	nonlocal	return	
as	def	finally	import	not	try	

1.3 Main Built-in types

Main built-in types in Python

Cat.	Type	Constructor	Create	Mut.	Index
Num.	int	int()	1	no	✗
Num.	float	float()	1.0	no	✗
Num.	complex	complex()	1+2j	no	✗
Seq.	list	list()	[1, "a"]	yes	int
Seq.	tuple	tuple()	(1, "a") or 1, "a"	no	int
Seq.	range	range()	range(bg[,end][,stp])	no	int
Set	set	set()	{0, 1}	yes	✗
Set	frozenset	frozenset()	frozenset(sequence)	no	✗
Txt seq.	str	str()	' ', " " or """ """	no	int
Map	dict	dict()	{"k1":1, "k2":"a" }	yes	key
Other	bool	bool()	True/False	no	✗
Other	module	✗	import	yes	✗
Other	class	✗	class	yes	✗
Other	method	✗	def	yes	✗
+ np	ndarray	array()	array(sequence)	yes	int

1.4 Indexing and slicing



1.5 Arithmetic operators

Operators applied to `complex`, `float` and `integers` objects			
	Operation	**Operator**	**Description**
	Addition	`a + b`	Sum of **a** and **b**
	Subtraction	`a - b`	Subtraction of **a** and **b**
	Multiplication	`a * b`	Multiplication of **a** by **b**
	Division	`a / b`	Division of **a** by **b**
	Power	`a**b` ou `pow(a, b)`	**a** to the power of b
	Modulus	`abs(a)`	Modulus of **a**
*	Division - remainder	`a%b`	Remainder of **a** / **b**
*	Division - integer part	`a//b`	Integer part of **a** / **b**
*	Full division	`divmod(a, b)`	Return the tuple `(a//b, a%b)`
	Conjugate	`a.conjugate()`	Return the conjugate of **a**
*	Integer conversion	`int(a)`	Convert **a** to `int` (truncation)
*	Float conversion	`float(a)`	Convert **a** to `float`
	Complex conversion	`complex(a)`	Convert **a** to `complex`
* These operations are not supported by `complex` objects			

1.6 Comparison, membership, identity and logical operators

Relational operators : Compare the values on either side of them		
Equality/Difference	`==` et `!=`	–
Superiority/Inferiority	`>` / `<` *ou* `>=` / `<=`	–
Logical operators : Classical logical comparisons		
Logical .AND.	`and`	–
Logical .OR.	`or`	–
Logical .NOT.	`not`	Inverts the adjacent operator
Membership operators : Test for membership in sequence		
Included	*var* `in` *seq*	`True if` *var* `in` *seq*, `False` otherwise
Excluded	*var* `not in` *seq*	`True if` *var* `not in` *seq*, `False` otherwise
Identity operators : Compare memory location of objects		
Equality	*a* `is` *b*	`True if` `id`(a) == `id`(b), `False` otherwise
Difference	*a* `is not` *b*	`True if` `id`(a) != `id`(b), `False` otherwise

1.7 Operation on sequences

Operator	**Description**
Common operation on sequences	
`x in s`	`True` if `x` is in `s`, `False` otherwise
`x not in s`	`False` if `x` is in `s`, `True` otherwise
`s + t`	Concatenate `s` and `t`
`s * n`	Repeat `s` `n` times
`s[i]`	i-th element of `s` (starting from 0)
`s[i:j:k]`	Slice of `s` from `i` to `j` in step of `k`
`len(s)`	Length of `s`
`min(s)`	Smallest element of `s`
`max(s)`	Largest element `s`
`s.count(x)`	Number of occurences of `x` in `s`
`s.index(x)`	Index of the 1st occurence of `x` in `s`
Operations on mutable sequences	
`s[i] = x`	Element `i` of `s` replaced by `x`
`s[i:j] = t`	Slice of `s` from `i` to `j` replaced by the iterable `t` of the same size
`del s[i:j:k]`	Delete the slice of elements - Similar to `s[i:j:k] = []`
`s.append(x)`	Append `x` to the end of the sequence
`s.clear()`	Delete all elements of `s`
`s.copy()`	Create a superficial copy of `s` (similar to `s[:]`)
`s.extend(t)`	Extend `s` with the iterable `t` (similar to `s += t`)
`s *= n`	Repeat `s` `n` time and update its content
`s.insert(i, x)`	Insert `x` in `s` at index `i`
`s.pop(i)`	Get the i-th element and delete it from `s`
`s.remove(x)`	Delete the first i-th element of `s` satisfying the condition `s[i] == x`
`s.reverse()`	Reverse the elements of `s` in place

1.8 File modes

File modes		
`r`	Read-only	Read access only. Default value.
`w`	Write-only	Write access only. Existing file with same name is overwritten.
`a`	Append	Add at the end of the file. Create a new file if not existing.
`r+`	Read and write	Read and write access.
`b`	Binary mode	Used for binary files.

2 Special functions

Class management		
`self = MyClass(args)`	`__new__(cls, args)`	(Constructor)
`self = MyClass(args)`	`__init__(self, args)`	(Init. args)
`del self`	`__del__(self)`	(Destructor)
`self(args)`	`__call__(self, args)`	(Make self callable)
✗	`__slots__(self)`	(To save memory)

Class representation		
`repr(self)`	`__repr__(self)`	
`str(self)`/`print(self)`	`__str__(self)`	
`unicode(self)`	`__unicode__(self)`	(Python 2)
`bytes(self)`	`__bytes__(self)`	(Python 3)
`format(self, spec)`	`__format__(self, spec)`	
`hash(self)`	`__hash__(self)`	
`dir(self)`	`__dir__(self)`	(List attributes)

Reflection		
`isinstance(inst, class)`	`__instancecheck__(self, inst)`	
`issubclass(sub, class)`	`__subclasscheck__(self, sub)`	
`issubclass(sub, class)`	`__subclasshook__(self, sub)`	(For abstract class)

Context manager	
`with self as x:`	`__enter__(self)`
`with self as x:`	`__exit__(self, exc, val, trace)`

Attributes		
`self.name`	`__getattribute__(self, name)`	(Unconditionally)
`self.name`	`__getattr__(self, name)`	(If name doesn't exist)
`self.name = val`	`__setattr__(self, name, val)`	
`del self.name`	`__delattr__(self, name)`	

Properties	
`self.property`	`__get__(self, property)`
`self.property = val`	`__set__(self, property, value)`
`del self.property`	`__delete__(self, property)`

Sequences		
`self[key]`	`__getitem__(self, key)`	
`self[key] = value`	`__setitem__(self, key, value)`	
`del self[key]`	`__delitem__(self, key)`	
`self[key]`	`__missing__(self, key)`	(If key doesn't exist)
`len(self)`	`__len__(self)`	
`value in self`	`__contains__(self, value)`	

Iterators		
`iter(self)`	`__iter__(self)`	
`next(self)`	`__next__(self)`	
`reversed(self)`	`__reversed__(self)`	(Reversed iterator)

Module copy	
`copy.copy(self)`	`__copy__(self)`
`copy.deepcopy(self)`	`__deepcopy__(self)`

Module pickle	
`pickle.dump(file, self)`	`__getstate__(self)`
`pickle.dump(file, self)`	`__reduce__(self)`
`pickle.dump(file, self)`	`__reduce_ex__(self)`
`x = pickle.load(file)`	`__getnewargs__(self)`
`x = pickle.load(file)`	`__getinitargs__(self)`
`x = pickle.load(file)`	`__setstate__(self)`

Module math	
`math.floor(self)`	`__floor__(self)`
`math.ceil(self)`	`__ceil__(self)`
`math.trunc(self)`	`__trunc__(self)`

Module sys	
`sys.sizeof(self)`	`__sizeof__(self)`

Binary operators *(i.e. between 2)*

Operator	Method	Reverse Method
+	`__add__(self, other)`	`__radd__()`
-	`__sub__(self, other)`	`__rsub__()`
*	`__mul__(self, other)`	`__rmul__()`
//	`__floordiv__(self, other)`	`__rfloordiv__()`
/	`__div__(self, other)` (Python 2)	`__rdiv__()`
/	`__truediv__(self, other)` (Python 3)	`__rtruediv__()`
%	`__mod__(self, other)`	`__rmod__()`
`divmod()`	`__divmod__(self, other)`	`__rdivmod__()`
**	`__pow__(self, other[, modulo])`	`__rpow__()`
<<	`__lshift__(self, other)`	`__rlshift__()`
>>	`__rshift__(self, other)`	`__rrshift__()`
&	`__and__(self, other)`	`__rand__()`
^	`__xor__(self, other)`	`__rxor__()`
|	`__or__(self, other)`	`__ror__()`

Comparison operators

`all operators`	`__cmp__(self, other)`	(Only in Python 2)
<	`__lt__(self, other)`	
<=	`__le__(self, other)`	
==	`__eq__(self, other)`	
!=	`__ne__(self, other)`	
>=	`__ge__(self, other)`	
>	`__gt__(self, other)`	

Assignment operators

+=	`__iadd__(self, other)`
-=	`__isub__(self, other)`
*=	`__imul__(self, other)`
/=	`__idiv__(self, other)`
//=	`__ifloordiv__(self, other)`
%=	`__imod__(self, other)`
**=	`__ipow__(self, other[, modulo])`
<<=	`__ilshift__(self, other)`
>>=	`__irshift__(self, other)`
&=	`__iand__(self, other)`
^=	`__ixor__(self, other)`
|=	`__ior__(self, other)`

Unary operators and functions *(i.e. applied to 1)*

-self	`__neg__(self)`	
+self	`__pos__(self)`	
abs(self)	`__abs__(self)`	
~self	`__invert__(self)`	
complex(self)	`__complex__(self)`	
int(self)	`__int__(self)`	
round(self, n)	`__round__(self, [n])`	
float(self)	`__float__(self)`	
oct(self)	`__oct__(self)`	
hex(self)	`__hex__(self)`	
bool(self)	`__bool__(self)`	(`__nonzero__()` in Python 2)
x[self]	`__index__(self)`	

3 Numpy

3.1 Numpy mathematical functions

To complete : https://docs.scipy.org/doc/numpy/reference/routines.math.html

Main Numpy mathematical functions	
Function	**Description**
Trigonometric functions	
`sin(x)` / `cos(x)` / `tan(x)`	Sine, cosine, tangent element-wise
`arcsin(x)` / `arccos(x)` / `arctan(x)`	Inverse sine, cosine, tangent element-wise
Hyperbolic functions	
`sinh(x)` / `cosh(x)` / `tanh(x)`	Hyperbolic sin, cos, tan element-wise
`arcsinh(x)` / `arccosh(x)` / `arctanh(x)`	Inverse hyperbolic sin, cos, tan element-wise
Exponential and logarithmic functions	
`exp(x)` / `log(x)`	Exponential and natural logarithm element-wise
`log10(x)` / `log2(x)`	Base 10 and 2 logarithm element-wise
Handling complex numbers	
`angle(z)`	Angle of the complex argument

3.2 Numpy array creation routines

Main numpy array creation functions		
Python	**Description**	**Matlab**
Arrays of ones and zeros		
`empty(shape[, dtype, order])`	New array of given shape and type without initialization	×
`empty_like(a[, dtype, order])`	New empty array with same shape as the array a	×
`zeros(shape[, dtype, order])`	New array of given shape and size filled with 0	zeros
`zeros_like(a[, dtype, order])`	New zeros array with same shape as the array a	×
`ones(shape[, dtype, order])`	New array of given shape and size filled with 1	ones
`ones_like(a[, dtype, order])`	New ones array with same shape as the array a	×
`full(shape, val[, dtype, order])`	New array of given shape and size filled *val*	ones
`full_like(a[, dtype, order])`	New full array with same shape as the array a	×
`eye([N[, M, k, dtype])`	Array with ones on a diagonal and zeros elsewhere	×
`identity(N[, dtype])`	Identity array (special case of `eye()`)	*eye()*
Arrays from existing data		
`array(obj[, dtype, ...])`	Create a new array	[1 2 ; 3 4]
`asarray(a[, dtype, order])`	Convert a to an array (not necessary a copy)	×
`ascontiguousarray(a[, dtype])`	Convert a to a contiguous array in memory (C order)	×
`copy(a[, order])`	Return an array copy of a	A = B
`fromfile/fromfunction/fromiter...`	Collection of function to build arrays from objects	
Arrays from numerical ranges		
`arange([start,] stop[, step, ...])`	Evenly spaced values over a given interval	*start:step:stop*
`linspace([start, stop[, num, ...])`	Evenly spaced values over a given interval	*linspace*
`logspace([start, stop[, num, ...])`	Evenly spaced values over a given interval on log scale	*logspace*
`geomspace([start, stop[, num, ...])`	Same as logspace but with geometric progression	
`meshgrid(*xi, **kwargs)`	Create array from coordinate vectors	
`mgrid/ogrid`	Dense mutlidimensional meshgrid	
Building classical matrices		
`diag(d[, k])`	Extract a diagonal or construct a diagonal array	*diag()*
`diagflat(d[, k])`	2D array with the flattened input as diagonal	
`tri([N[, M, k, dtype])`	Tridiagonal array	×
`tril(m[, k])`	Lower triangular array	*tril()*
`triu(m[, k])`	Upper triangular array	*triu()*
`vander(x[, N, increasing])`	Vandermonde array	*vander()*

3.3 Numpy array attributes

Main Numpy array attributes	
Attribute	**Description**
Memory layout	
ndarray.flags	Information about the memory layout of the array
ndarray.shape	Tuple of array dimensions
ndarray.strides	Tuple of bytes to step in each dimension when traversing an array
ndarray.ndim	Number of array dimensions
ndarray.data	Python buffer object pointing to the start of the array's data
ndarray.size	Number of elements in the array
ndarray.itemsize	Length of one array element in bytes
ndarray.nbytes	Total bytes consumed by the elements of the array
ndarray.base	Base object if memory is from some other object
Other attributes	
ndarray.dtype	Data-type of the array's elements
ndarray.T	Transposed array
ndarray.real	Real part of the array
ndarray.imag	Imaginary part of the array
ndarray.flat	1-D iterator over the array

3.4 Numpy array methods

Main Numpy array methods	
Method	**Description**
Array conversion	
ndarray.item(*args)	Return a standard Python scalar that is a copy an element of an array
ndarray.tolist()	Return the array as a (possibly nested) list
ndarray.itemset(*args)	Insert scalar into an array (scalar is cast to array's dtype, if possible)
ndarray.tostring([order])	Construct Python bytes containing the raw data bytes in the array
ndarray.tobytes([order])	Construct Python bytes containing the raw data bytes in the array
ndarray.tofile(fid[, sep, format])	Write array to a file as text or binary (default)
ndarray.dump(file)	Dump a pickle of the array to the specified file
ndarray.dumps()	Returns the pickle of the array as a string
ndarray.astype(dtype[, order, casting, ...])	Copy of the array, cast to a specified type
ndarray.byteswap(inplace)	Swap the bytes of the array elements
ndarray.copy([order])	Return a copy of the array
ndarray.view([dtype, type])	New view of array with the same data
ndarray.getfield(dtype[, offset])	Returns a field of the given array as a certain type
ndarray.setflags([write, align, uic])	Set the three array flags
ndarray.fill(value)	Fill the array with a scalar value
Shape manipulation	
ndarray.reshape(shape[, order])	Return array with a new shape
ndarray.resize(new_shape)	Change shape and size of array in-place
ndarray.transpose(*axes)	Return a view of the array with axes transposed
ndarray.swapaxes(axis1, axis2)	Return a view of the array with axis1 and axis2 interchanged
ndarray.flatten([order])	Return a copy of the array collapsed into one dimension
ndarray.ravel([order])	Return a flattened array
ndarray.squeeze([axis])	Remove single-dimensional entries from the shape of a
Item selection and manipulation	
ndarray.take(indices[, axis, out, mode])	Return an array formed from the elements of a at the given indices
ndarray.put(indices, values[, mode])	Set a.flat[n] = values[n] for all n in indices
ndarray.repeat(repeats[, axis])	Repeat elements of an array
ndarray.choose(choices[, out, mode])	Use an index array to construct a new array from a set of choices
ndarray.sort([axis, kind, order])	Sort an array, in-place
ndarray.argsort([axis, kind, order])	Return the indices that would sort this array
ndarray.partition(kth[, axis, kind, order])	Rearranges the elements in the array in such a way that value of the element in kth position is in the position it would be in a sorted array
ndarray.argpartition(kth[, axis, kind, order])	Return the indices that would partition this array
ndarray.searchsorted(v[, side, sorter])	Find indices where elements of v should be inserted in a to maintain order
ndarray.nonzero()	Return the indices of the elements that are non-zero
ndarray.compress(condition[, axis, out])	Return selected slices of this array along given axis
ndarray.diagonal([offset, axis1, axis2])	Return specified diagonals
Calculation	
ndarray.argmin([axis, out])	Return indices of the minimum values along the given axis of a
ndarray.argmax([axis, out])	Return indices of the maximum values along the given axis
ndarray.min([axis, out, keepdims])	Return the minimum along a given axis
ndarray.max([axis, out, keepdims])	Return the maximum along a given axis
ndarray.ptp([axis, out])	Peak to peak (maximum - minimum) value along a given axis
ndarray.clip([min, max, out])	Return an array whose values are limited to [min, max]
ndarray.conj()	Complex-conjugate all elements
ndarray.round([decimals, out])	Return a with each element rounded to the given number of decimals
ndarray.trace([offset, axis1, axis2, dtype, out])	Return the sum along diagonals of the array
ndarray.sum([axis, dtype, out, keepdims])	Return the sum of the array elements over the given axis
ndarray.cumsum([axis, dtype, out])	Return the cumulative sum of the elements along the given axis
ndarray.mean([axis, dtype, out, keepdims])	Return the average of the array elements along given axis
ndarray.var([axis, dtype, out, ddof, keepdims])	Return the variance of the array elements, along given axis
ndarray.std([axis, dtype, out, ddof, keepdims])	Return the standard deviation of the array elements along given axis
ndarray.prod([axis, dtype, out, keepdims])	Return the product of the array elements over the given axis
ndarray.cumprod([axis, dtype, out])	Return the cumulative product of the elements along the given axis
ndarray.all([axis, out, keepdims])	Return True if all elements evaluate to True
ndarray.any([axis, out, keepdims])	Return True if any of the elements of a evaluate to True

3.5 Numpy array manipulation routines

Main array manipulation routines	
Method	**Description**
Joining arrays	
concatenate((a1, a2, ...)[, axis])	Join a sequence of arrays along an existing axis
stack(arrays[, axis])	Join a sequence of arrays along a new axis
column_stack(tup)	Stack 1-D arrays as columns into a 2-D array
dstack(tup)	Stack arrays in sequence depth wise (along third axis)
hstack(tup)	Stack arrays in sequence horizontally (column wise)
vstack(tup)	Stack arrays in sequence vertically (row wise)
Splitting arrays	
split(ary, indices_or_sections[, axis])	Split an array into multiple sub-arrays
array_split(ary, indices_or_sections[, axis])	Split an array into multiple sub-arrays
dsplit(ary, indices_or_sections)	Split array into multiple sub-arrays along the 3rd axis (depth)
hsplit(ary, indices_or_sections)	Split an array into multiple sub-arrays horizontally (column-wise)
vsplit(ary, indices_or_sections)	Split an array into multiple sub-arrays vertically (row-wise)
Tiling arrays	
tile(A, reps)	Construct an array by repeating A the number of times given by reps
repeat(a, repeats[, axis])	Repeat elements of an array
Adding and removing elements	
delete(arr, obj[, axis])	Return a new array with sub-arrays along an axis deleted
insert(arr, obj, values[, axis])	Insert values along the given axis before the given indices
append(arr, values[, axis])	Append values to the end of an array
resize(a, new_shape)	Return a new array with the specified shape
trim_zeros(filt[, trim])	Trim the leading and/or trailing zeros from a 1-D array or sequence
unique(ar[, return_index, return_inverse, ...])	Find the unique elements of an array
Rearranging elements	
flip(m, axis)	Reverse the order of elements in an array along the given axis
fliplr(m)	Flip array in the left/right direction
flipud(m)	Flip array in the up/down direction
reshape(a, newshape[, order])	Gives a new shape to an array without changing its data
roll(a, shift[, axis])	Roll array elements along a given axis
rot90(m[, k, axes])	Rotate an array by 90 degrees in the plane specified by axes

3.6 Numpy linear algebra

Numpy linear algebra functions	
Function	**Description**
Matrix and vector products	
`dot`	Dot product of two arrays
`linalg.multi_dot`	Dot product of two or more arrays in a single function call
`vdot`	Dot product of two vectors
`inner`	Inner product of two arrays
`outer`	Outer product of two arrays
`matmul`	Matrix product of two arrays
`tensordot`	Tensor dot product along specified axes for arrays >= 1-D
`einsum`	Einstein summation convention on the operands
`linalg.matrix_power`	Square matrix to the (integer) power n
`kron`	Kronecker product of two arrays
Decomposition	
`linalg.choleski`	Choleski decomposition
`linalg.qr`	Compute the qr factorization of a matrix
`linalg.svd`	Singular Value Decomposition
Matrix eigenvalues	
`linalg.eig`	Eigenvalues and right eigenvectors of a square array
`linalg.eigh`	Eigenvalues and eigenvectors of a Hermitian or symmetric matrix
`linalg.eigvals`	Eigenvalues of a general matrix
`linalg.eigvalsh`	Compute the eigenvalues of a Hermitian or real symmetric matrix
Norms and other numbers	
`linalg.norm`	Matrix or vector norm
`linalg.cond`	Compute the condition number of a matrix
`linalg.det`	Compute the determinant of an array
`linalg.matrix_rank`	Return matrix rank of array using SVD method
`linalg.slogdet`	Compute the sign and (natural) logarithm of the determinant of an array
`trace`	Return the sum along diagonals of the array
Solving equations and inverting matrices	
`linalg.solve`	Solve a linear matrix equation, or system of linear scalar equations
`linalg.tensorsolve`	Solve the tensor equation $ax = b$ for x
`linalg.lstsq`	Return the least-squares solution to a linear matrix equation
`linalg.inv`	Compute the (multiplicative) inverse of a matrix
`linalg.pinv`	Compute the (Moore-Penrose) pseudo-inverse of a matrix
`linalg.tensorinv`	Compute the inverse of an N-dimensional array

3.7 Numpy load and save methods

Numpy functions for File I/O	
Method	**Description**
`loadtxt()/savetxt()`	Read/Save an array to a text file
`load()/save()`	Read/Save an array to a `.npy` binary file
`load()/savez()`	Save several arrays into an `.npz` uncompressed archive
`load()/savez_compressed()`	Save several arrays into an `.npz` compressed archive

Saving/loading time for an array of 10^6 random elements		
Function	**Execution time**	**Disk usage**
`loadtxt()/savetxt()`	900ms/700ms	24 Mo
`load()/save()`	2ms/25ms	7.7 Mo
`load()/savez()`	45μs/45ms	7.7 Mo
`load()/savez_compressed()`	45μs/325ms	7.2 Mo (7.4 Ko for np.zeros)

4 Scipy subpackages

Scipy subpackages	
Subpackage	**Description**
`cluster`	Clustering algorithms
`constants`	Physical and mathematical constants
`fftpack`	Fast Fourier Transform routines
`integrate`	Integration and ordinary differential equation solvers
`interpolate`	Interpolation and smoothing splines
`io`	Input and Output
`linalg`	Linear algebra
`ndimage`	N-dimensional image processing
`odr`	Orthogonal distance regression
`optimize`	Optimization and root-finding routines
`signal`	Signal processing
`sparse`	Sparse matrices and associated routines
`spatial`	Spatial data structures and algorithms
`special`	Special functions
`stats`	Statistical distributions and functions

5 Matplotlib.pyplot summary

Main Pyplot Functions	
Function	**Description**
Figure initialization	
`fig = plt.figure(figsize=(6, 3))`	Initialize figure
`fig.savefig("out.png")`	Save figure to png image
`fig, axes = plt.subplots(3, 2, figsize=(8, 8))`	Figure with 3 × 2 array of axes
`fig, ax = plt.subplots(figsize=(8, 4))`	Figure default 111 subplot
Figure properties	
`fig.suptitle("Title")`	Figure title
`fig.tight_layout()`	Adjust subplots to fit into figure
`ax.set_xlabel("x")`	Set xlabel
`ax.set_ylabel("y")`	Set ylabel
`ax.set_xlim(1, 2)`	Sets x limits
`ax.set_ylim(3, 4)`	Sets y limits
`ax.set_title("MyTitle")`	Sets the axis title
`ax.set(xlabel="x", ...)`	Set multiple parameters at once
`ax.legend(loc="upper center")`	Activate legend
`ax.grid(True, which="both")`	Activate grid
Some plotting routines	
`ax.plot(x, y)`	Plots lines
`ax.semilogx(x, y)`	Semilog plot
`ax.semilogy(x, y)`	Semilog plot
`ax.loglog(x, y)`	Loglog plot
`ax.scatter(x, y)`	Scatter plot
`ax.pcolormesh(xx, yy, zz)`	Fast colormesh
`ax.colormesh(xx, yy, zz)`	Colormesh
`ax.contour(xx, yy, zz)`	Contour lines
`ax.contourf(xx, yy, zz)`	Filled contours
`ax.imshow(xx)`	Show image
`ax.text(x, y, string, fontsize=12, color="m")`	Write text

6 Other modules

6.1 os module : main functions

Main functions of the os module	
Method	**Description**
`os.uname()`	Return information identifying the current OS
`os.listdir(path)`	Return a list of the entries in the directory given by path
`os.mkdir(path)`	Create a directory
`os.makedirs(path)`	Recursively create a directory
`os.rmdir(path)`	Remove a directory
`os.removedirs(path)`	Recursively remove directories
`os.rename(src, dest)`	Rename a directory or a file
`os.remove(path)`	Remove a file
`os.system()`	Execute an external command
`os.getcurdir (/getcwd)`	Returns the current (/working) directory
`os.chdir()`	Change working directory

6.2 re modules : regular expressions

Main metacharacters and classes	
Metachar	**Description**
`?`	Matches the preceding element 0 or 1 time
`+`	Matches the preceding element 1 or more times
`*`	Matches the preceding element 0, 1, or more times
`.`	Matches any single character
`^`	Matches the starting position, or opposite of
`$`	Matches the ending position
`[ ]`	Matches a single character that is contained within the brackets
`( )`	Marked subexpression (also known as *block* or *capturing group*)
`|`	Matches either the expression before or the expression after the operator
`{m, n}`	Matches the preceding element at least m and not more than n times
`\d`	Matches any decimal digit : equivalent to the class [0-9]
`\D`	Matches any non digit character : equivalent to the class [^0-9]
`\s`	Matches any whitespace character
`\S`	Matches any non-whitespace character
`\w`	Matches any alphanumeric character : equivalent to [a-zA-Z0-9_]
`\W`	Matches any non-alphanumeric character ; equivalent to [^a-zA-Z0-9_]
`\`	Escape character

7 Matlab equivalences

7.1 Banana skins for Matlab users

Also see this link : https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html

	Python	Matlab
Script file extension	.py	.m
Line continuation	\	...
Comment	#	%
First element index	A[0]	A(1)
Modulus (returns remainder)	%	mod(a, b)
Floor division	//	floor(a/b)
Last element index	A[-1]	A(end)
Slicing	A[0:5] or A[:5]	A(1:5)
Display to standard output	print(x)	disp(x)
Help on a function	help(func)	help func
Mutability	B=A.copy() or B=list(A)	B=A
Import library functions	import package	Must be in MATLABPATH
Print source code	function??	×
Difference	!=	~=
Logical and	A and B	A && B
Logical or	A or B	A || B

7.2 Numpy array manipulation routines

Main array manipulation routines	
Method	**Description**
Joining arrays	
concatenate((a1, a2, ...)[, axis])	Join a sequence of arrays along an existing axis
stack(arrays[, axis])	Join a sequence of arrays along a new axis
column_stack(tup)	Stack 1-D arrays as columns into a 2-D array
dstack(tup)	Stack arrays in sequence depth wise (along third axis)
hstack(tup)	Stack arrays in sequence horizontally (column wise)
vstack(tup)	Stack arrays in sequence vertically (row wise)
Splitting arrays	
split(ary, indices_or_sections[, axis])	Split an array into multiple sub-arrays
array_split(ary, indices_or_sections[, axis])	Split an array into multiple sub-arrays
dsplit(ary, indices_or_sections)	Split array into multiple sub-arrays along the 3rd axis (depth)
hsplit(ary, indices_or_sections)	Split an array into multiple sub-arrays horizontally (column-wise)
vsplit(ary, indices_or_sections)	Split an array into multiple sub-arrays vertically (row-wise)
Tiling arrays	
tile(A, reps)	Construct an array by repeating A the number of times given by reps
repeat(a, repeats[, axis])	Repeat elements of an array
Adding and removing elements	
delete(arr, obj[, axis])	Return a new array with sub-arrays along an axis deleted
insert(arr, obj, values[, axis])	Insert values along the given axis before the given indices
append(arr, values[, axis])	Append values to the end of an array
resize(a, new_shape)	Return a new array with the specified shape
trim_zeros(filt[, trim])	Trim the leading and/or trailing zeros from a 1-D array or sequence
unique(ar[, return_index, return_inverse, ...])	Find the unique elements of an array
Rearranging elements	
flip(m, axis)	Reverse the order of elements in an array along the given axis
fliplr(m)	Flip array in the left/right direction
flipud(m)	Flip array in the up/down direction
reshape(a, newshape[, order])	Gives a new shape to an array without changing its data
roll(a, shift[, axis])	Roll array elements along a given axis
rot90(m[, k, axes])	Rotate an array by 90 degrees in the plane specified by axes