`python` Intermediate

`NumPy` introduction and `arrays`

Mauricio Sevilla

email= j.sevillam@uniandes.edu.co

18.03.19

We have already explored several features of python itself, some structures and build in functions.

Now, we are going to learn how to use one of the most important libraries.

The reason why `NumPy` is so important is because it has a data type very efficient, the `numpy.array` s and almost all the *big* libraries used nowadays are based on them.

For further information of the package, link (http://www.numpy.org/).

Let us exlore the data first, and then, sumarize the principal features.

In [19]:
```
import numpy as np
```

importing `numpy` with the alias `np` is not mandatory but standard. All the official documentation and therefore the forums and examples use this.

In [20]:
```
a=np.array([1,2.0,3])
```

In [21]:
```
print(type(a),type(a[0]))
```

```
<class 'numpy.ndarray'> <class 'numpy.float64'>
```

In [22]:
```
b=[5,4,3]
```

In [23]:
```
a+b
```

Out[23]:
```
array([6., 6., 6.])
```

In [24]:
```
a-b
```

Out[24]:
```
array([-4., -2.,  0.])
```

In [25]:
```
a/b
```

Out[25]:
```
array([0.2, 0.5, 1. ])
```

In [26]:
```
a*b
```

Out[26]:
```
array([5., 8., 9.])
```

In [27]:
```
a**b
```

Out[27]:
```
array([ 1., 16., 27.])
```

In [28]:
```
a//b
```

Out[28]:
```
array([0., 0., 1.])
```

In [29]:
```
a%b
```

Out[29]:
```
array([1., 2., 0.])
```

As you can see, as the operations are done component by component compared with the `list` s, more operations are allowed.

In [30]:
```
c=np.array([5,6])
```

In [31]:
```
a+c
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-31-e200917bc55f> in <module>
----> 1 a+c

ValueError: operands could not be broadcast together with shapes (3,) (2,)
```

Explore the rest of them! and you'll see that dimensions here, are very important when working with `array` s

In [32]:
```
a1=[1]
b1=np.array([1])
print(2*a1,2*b1)
```

```
[1, 1] [2]
```

The result of the multiplication by a number is completely different!!.

How can we add a new value?, when working with `list` s, we could add another list or use the `append` method.

On `numpy.array` s we cannot *add* an extra element, but we can use something like the `append` method, but now it is a function!,

Do you remember the difference between the two notations?

In [33]:
```
print(c,type(c))
```

```
[5 6] <class 'numpy.ndarray'>
```

In [34]:
```
c=np.append(c,1)
```

In [35]:
```
c
```

Out[35]:
```
array([5, 6, 1])
```

We just save on `c` the result of taking the array `c` and an array with the `1` and appending them.

It means that, we can use the append mehtod to concatenate arrays, such as the sum does for the lists.

In [36]:
```
print(b,type(b))
```

```
[5, 4, 3] <class 'list'>
```

In [37]:
```
b=np.append(b,1)
```

In [38]:
```
b
```

Out[38]:
```
array([5, 4, 3, 1])
```

It is important that after the operation `b` becomes an `numpy.array`.

Even, different `array` s can be used on the function `numpy.append`

In [39]:
```
print(a,b,np.append(a,b))
```

```
[1. 2. 3.] [5 4 3 1] [1. 2. 3. 5. 4. 3. 1.]
```

So that, they can be assigned to a new variable.

In [40]:
```
d=np.append(a,b)
```

In [41]:
```
print(d)
```

```
[1. 2. 3. 5. 4. 3. 1.]
```

To know the possible functions and variables of `numpy` (Which are a lot!), we can use `dir` as we already have discussed.

In [42]:
```
dir(np)
```

Out[42]:
```
['ALLOW_THREADS',
 'AxisError',
 'BUFSIZE',
 'CLIP',
 'ComplexWarning',
 'DataSource',
 'ERR_CALL',
 'ERR_DEFAULT',
 'ERR_IGNORE',
 'ERR_LOG',
 'ERR_PRINT',
 'ERR_RAISE',
 'ERR_WARN',
 'FLOATING_POINT_SUPPORT',
 'FPE_DIVIDEBYZERO',
 'FPE_INVALID',
 'FPE_OVERFLOW',
 'FPE_UNDERFLOW',
 'False_',
 'Inf',
 'Infinity',
 'MAXDIMS',
 'MAY_SHARE_BOUNDS',
 'MAY_SHARE_EXACT',
 'MachAr',
 'ModuleDeprecationWarning',
 'NAN',
 'NINF',
 'NZERO',
 'NaN',
 'PINF',
 'PZERO',
 'PackageLoader',
 'RAISE',
 'RankWarning',
```

```
'SHIFT_DIVIDEBYZERO',
'SHIFT_INVALID',
'SHIFT_OVERFLOW',
'SHIFT_UNDERFLOW',
'ScalarType',
'Tester',
'TooHardError',
'True_',
'UFUNC_BUFSIZE_DEFAULT',
'UFUNC_PYVALS_NAME',
'VisibleDeprecationWarning',
'WRAP',
'_NoValue',
'__NUMPY_SETUP__',
'__all__',
'__builtins__',
'__cached__',
'__config__',
'__doc__',
'__file__',
'__git_revision__',
'__loader__',
'__name__',
'__package__',
'__path__',
'__spec__',
'__version__',
'_distributor_init',
'_globals',
'_import_tools',
'_mat',
'abs',
'absolute',
'absolute_import',
'add',
'add_docstring',
'add_newdoc',
'add_newdoc_ufunc',
```

```
'add_newdocs',
'alen',
'all',
'allclose',
'alltrue',
'amax',
'amin',
'angle',
'any',
'append',
'apply_along_axis',
'apply_over_axes',
'arange',
'arccos',
'arccosh',
'arcsin',
'arcsinh',
'arctan',
'arctan2',
'arctanh',
'argmax',
'argmin',
'argpartition',
'argsort',
'argwhere',
'around',
'array',
'array2string',
'array_equal',
'array_equiv',
'array_repr',
'array_split',
'array_str',
'asanyarray',
'asarray',
'asarray_chkfinite',
'ascontiguousarray',
'asfarray',
```

```
'asfortranarray',
'asmatrix',
'asscalar',
'atleast_1d',
'atleast_2d',
'atleast_3d',
'average',
'bartlett',
'base_repr',
'binary_repr',
'bincount',
'bitwise_and',
'bitwise_not',
'bitwise_or',
'bitwise_xor',
'blackman',
'block',
'bmat',
'bool',
'bool8',
'bool_',
'broadcast',
'broadcast_arrays',
'broadcast_to',
'busday_count',
'busday_offset',
'busdaycalendar',
'byte',
'byte_bounds',
'bytes0',
'bytes_',
'c_',
'can_cast',
'cast',
'cbrt',
'cdouble',
'ceil',
'cfloat',
```

```
'char',
'character',
'chararray',
'choose',
'clip',
'clongdouble',
'clongfloat',
'column_stack',
'common_type',
'compare_chararrays',
'compat',
'complex',
'complex128',
'complex256',
'complex64',
'complex_',
'complexfloating',
'compress',
'concatenate',
'conj',
'conjugate',
'convolve',
'copy',
'copysign',
'copyto',
'core',
'corrcoef',
'correlate',
'cos',
'cosh',
'count_nonzero',
'cov',
'cross',
'csingle',
'ctypeslib',
'cumprod',
'cumproduct',
'cumsum',
```

```
'datetime64',
'datetime_as_string',
'datetime_data',
'deg2rad',
'degrees',
'delete',
'deprecate',
'deprecate_with_doc',
'diag',
'diag_indices',
'diag_indices_from',
'diagflat',
'diagonal',
'diff',
'digitize',
'disp',
'divide',
'division',
'divmod',
'dot',
'double',
'dsplit',
'dstack',
'dtype',
'e',
'ediff1d',
'einsum',
'einsum_path',
'emath',
'empty',
'empty_like',
'equal',
'errstate',
'euler_gamma',
'exp',
'exp2',
'expand_dims',
'expm1',
```

```
'extract',
'eye',
'fabs',
'fastCopyAndTranspose',
'fft',
'fill_diagonal',
'find_common_type',
'finfo',
'fix',
'flatiter',
'flatnonzero',
'flexible',
'flip',
'fliplr',
'flipud',
'float',
'float128',
'float16',
'float32',
'float64',
'float_',
'float_power',
'floating',
'floor',
'floor_divide',
'fmax',
'fmin',
'fmod',
'format_float_positional',
'format_float_scientific',
'format_parser',
'frexp',
'frombuffer',
'fromfile',
'fromfunction',
'fromiter',
'frompyfunc',
'fromregex',
```

```
'fromstring',
'full',
'full_like',
'fv',
'gcd',
'generic',
'genfromtxt',
'geomspace',
'get_array_wrap',
'get_include',
'get_printoptions',
'getbufsize',
'geterr',
'geterrcall',
'geterrobj',
'gradient',
'greater',
'greater_equal',
'half',
'hamming',
'hanning',
'heaviside',
'histogram',
'histogram2d',
'histogram_bin_edges',
'histogramdd',
'hsplit',
'hstack',
'hypot',
'i0',
'identity',
'iinfo',
'imag',
'in1d',
'index_exp',
'indices',
'inexact',
'inf',
```

```
'info',
'infty',
'inner',
'insert',
'int',
'int0',
'int16',
'int32',
'int64',
'int8',
'int_',
'int_asbuffer',
'intc',
'integer',
'interp',
'intersect1d',
'intp',
'invert',
'ipmt',
'irr',
'is_busday',
'isclose',
'iscomplex',
'iscomplexobj',
'isfinite',
'isfortran',
'isin',
'isinf',
'isnan',
'isnat',
'isneginf',
'isposinf',
'isreal',
'isrealobj',
'isscalar',
'issctype',
'issubclass_',
'issubdtype',
```

```
'issubsctype',
'iterable',
'ix_',
'kaiser',
'kron',
'lcm',
'ldexp',
'left_shift',
'less',
'less_equal',
'lexsort',
'lib',
'linalg',
'linspace',
'little_endian',
'load',
'loads',
'loadtxt',
'log',
'log10',
'log1p',
'log2',
'logaddexp',
'logaddexp2',
'logical_and',
'logical_not',
'logical_or',
'logical_xor',
'logspace',
'long',
'longcomplex',
'longdouble',
'longfloat',
'longlong',
'lookfor',
'ma',
'mafromtxt',
'mask_indices',
```

```
'mat',
'math',
'matmul',
'matrix',
'matrixlib',
'max',
'maximum',
'maximum_sctype',
'may_share_memory',
'mean',
'median',
'memmap',
'meshgrid',
'mgrid',
'min',
'min_scalar_type',
'minimum',
'mintypecode',
'mirr',
'mod',
'modf',
'moveaxis',
'msort',
'multiply',
'nan',
'nan_to_num',
'nanargmax',
'nanargmin',
'nancumprod',
'nancumsum',
'nanmax',
'nanmean',
'nanmedian',
'nanmin',
'nanpercentile',
'nanprod',
'nanquantile',
'nanstd',
```

```
'nansum',
'nanvar',
'nbytes',
'ndarray',
'ndenumerate',
'ndfromtxt',
'ndim',
'ndindex',
'nditer',
'negative',
'nested_iters',
'newaxis',
'nextafter',
'nonzero',
'not_equal',
'nper',
'npv',
'numarray',
'number',
'obj2sctype',
'object',
'object0',
'object_',
'ogrid',
'oldnumeric',
'ones',
'ones_like',
'outer',
'packbits',
'pad',
'partition',
'percentile',
'pi',
'piecewise',
'pkgload',
'place',
'pmt',
'poly',
```

```
'poly1d',
'polyadd',
'polyder',
'polydiv',
'polyfit',
'polyint',
'polymul',
'polynomial',
'polysub',
'polyval',
'positive',
'power',
'ppmt',
'print_function',
'printoptions',
'prod',
'product',
'promote_types',
'ptp',
'put',
'put_along_axis',
'putmask',
'pv',
'quantile',
'r_',
'rad2deg',
'radians',
'random',
'rank',
'rate',
'ravel',
'ravel_multi_index',
'real',
'real_if_close',
'rec',
'recarray',
'recfromcsv',
'recfromtxt',
```

```
'reciprocal',
'record',
'remainder',
'repeat',
'require',
'reshape',
'resize',
'result_type',
'right_shift',
'rint',
'roll',
'rollaxis',
'roots',
'rot90',
'round',
'round_',
'row_stack',
's_',
'safe_eval',
'save',
'savetxt',
'savez',
'savez_compressed',
'sctype2char',
'sctypeDict',
'sctypeNA',
'sctypes',
'searchsorted',
'select',
'set_numeric_ops',
'set_printoptions',
'set_string_function',
'setbufsize',
'setdiff1d',
'seterr',
'seterrcall',
'seterrobj',
'setxor1d',
```

```
'shape',
'shares_memory',
'short',
'show_config',
'sign',
'signbit',
'signedinteger',
'sin',
'sinc',
'single',
'singlecomplex',
'sinh',
'size',
'sometrue',
'sort',
'sort_complex',
'source',
'spacing',
'split',
'sqrt',
'square',
'squeeze',
'stack',
'std',
'str',
'str0',
'str_',
'string_',
'subtract',
'sum',
'swapaxes',
'sys',
'take',
'take_along_axis',
'tan',
'tanh',
'tensordot',
'test',
```

```
'testing',
'tile',
'timedelta64',
'trace',
'tracemalloc_domain',
'transpose',
'trapz',
'tri',
'tril',
'tril_indices',
'tril_indices_from',
'trim_zeros',
'triu',
'triu_indices',
'triu_indices_from',
'true_divide',
'trunc',
'typeDict',
'typeNA',
'typecodes',
'typename',
'ubyte',
'ufunc',
'uint',
'uint0',
'uint16',
'uint32',
'uint64',
'uint8',
'uintc',
'uintp',
'ulonglong',
'unicode',
'unicode_',
'union1d',
'unique',
'unpackbits',
'unravel_index',
```

In [43]:
```
np.argmax(d)
```

Out[43]: 3

In [44]:
```
print(d,np.argmax(d),d[np.argmax(d)])
```

```
[1. 2. 3. 5. 4. 3. 1.] 3 5.0
```

What if that number appears twice or more?

In [45]:
```
d=np.append(d,5)
```

In [46]:
```
print(np.argmax(d))
```

```
3
```

The function returns the argument of the first one.

`numpy` has also implemented some mathematical functions, but we have also seen the library `math`, let us compare them.

In [47]:
```
import math as m
```

In [48]:
```
print(m.cos(0),type(m.cos(0)))
```

```
1.0 <class 'float'>
```

As you can see, the function `math.cos` when recieves a number, returns a `float`. But what if we use a `list`.

In [49]:
```
list_test=[1,2,3]
```

In [50]:
```
print(m.cos(list_test))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-50-f12eb81b5249> in <module>
----> 1 print(m.cos(list_test))

TypeError: must be real number, not list
```

The function `math.cos` does not allow us to use `list`s as parameters, but what about `numpy` ?

In [51]:
```
print(np.cos(list_test))
```

```
[ 0.54030231 -0.41614684 -0.9899925 ]
```

if we save the result on a new variable,

what data type do you think is the result?

In [52]:
```
test_cos=np.cos(list_test)
```

In [53]:
```
print(type(test_cos))
```

```
<class 'numpy.ndarray'>
```

It also can be used with numbers,

In [54]:
```
print(np.cos(0),type(np.cos(0)))
```

```
1.0 <class 'numpy.float64'>
```

There are two functions which are special, because we are going to use them later a lot!

In [55]:
```
print(np.linspace(1,2,11))
```

```
[1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2. ]
```

In [56]:
```
print(np.arange(1,2.1,0.1))
```

```
[1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2. ]
```

The logic is different, when using `numpy.linspace` you need to use the starting point, ending point, and amount of points, while when working with `numpy.arange` , you use the initial point, final point (NOT INCLUDED!) and step.

Let see another example, look how powerful `numpy` is, on a previous homework we worked on sorting.

Imagine that we have an array with repeated values and unsorted, such as,

In [61]:
```
d=np.array([5,7,9,1,2,2,3,9,5,0,1])
```

The function `numpy.unique` , constucts a new array sorted an without repeated values,

In [62]:
```
np.unique(d)
```

Out[62]:
```
array([0, 1, 2, 3, 5, 7, 9])
```

You might take a look to the documentation link (https://docs.scipy.org/doc/numpy/user/basics.html)!!