## A Python Integer Is More Than Just an Integer

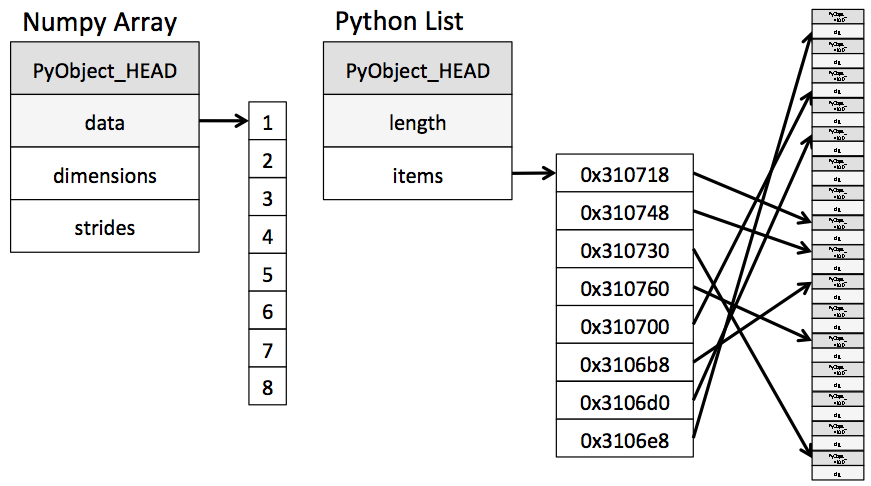
The standard Python implementation is written in C. This means that every Python object is simply a cleverly-disguised C structure, which contains not only its value, but other information as well. For example, when we define an integer in Python, such as x = 10000, x is not just a "raw" integer. It's actually a pointer to a compound C structure, which contains several values. Looking through the Python 3.4 source code, we find that the integer (long) type definition effectively looks like this (once the C macros are expanded):

A single integer in Python 3.4 actually contains four pieces:

* ob\_refcnt, a reference count that helps Python silently handle memory allocation and deallocation
* ob\_type, which encodes the type of the variable
* ob\_size, which specifies the size of the following data members
* ob\_digit, which contains the actual integer value that we expect the Python variable to represent.

This means that there is some overhead in storing an integer in Python as compared to an integer in a compiled language like C,

But this flexibility comes at a cost: to allow these flexible types, each item in the list must contain its own type info, reference count, and other information–that is, each item is a complete Python object. In the special case that all variables are of the same type, much of this information is redundant: it can be much more efficient to store data in a fixed-type array. The difference between a dynamic-type list and a fixed-type (NumPy-style) array is illustrated in the following figure:

****

At the implementation level, the array essentially contains a single pointer to one contiguous block of data. The Python list, on the other hand, contains a pointer to a block of pointers, each of which in turn points to a full Python object like the Python integer we saw earlier. Again, the advantage of the list is flexibility: because each list element is a full structure containing both data and type information, the list can be filled with data of any desired type. Fixed-type NumPy-style arrays lack this flexibility, but are much more efficient for storing and manipulating data.

**Datatypes:**

| **Data type** | **Description** |
| --- | --- |
| bool\_ | Boolean (True or False) stored as a byte |
| int\_ | Default integer type (same as C long; normally either int64 or int32) |
| intc | Identical to C int (normally int32 or int64) |
| intp | Integer used for indexing (same as C ssize\_t; normally either int32 or int64) |
| int8 | Byte (-128 to 127) |
| int16 | Integer (-32768 to 32767) |
| int32 | Integer (-2147483648 to 2147483647) |
| int64 | Integer (-9223372036854775808 to 9223372036854775807) |
| uint8 | Unsigned integer (0 to 255) |
| uint16 | Unsigned integer (0 to 65535) |
| uint32 | Unsigned integer (0 to 4294967295) |
| uint64 | Unsigned integer (0 to 18446744073709551615) |
| float\_ | Shorthand for float64. |
| float16 | Half precision float: sign bit, 5 bits exponent, 10 bits mantissa |
| float32 | Single precision float: sign bit, 8 bits exponent, 23 bits mantissa |
| float64 | Double precision float: sign bit, 11 bits exponent, 52 bits mantissa |
| complex\_ | Shorthand for complex128. |
| complex64 | Complex number, represented by two 32-bit floats |
| complex128 | Complex number, represented by two 64-bit floats |

**few categories of basic array manipulations here**:

* *Attributes of arrays*: Determining the size, shape, memory consumption, and data types of arrays
* *Indexing of arrays*: Getting and setting the value of individual array elements
* *Slicing of arrays*: Getting and setting smaller subarrays within a larger array
* *Reshaping of arrays*: Changing the shape of a given array
* *Joining and splitting of arrays*: Combining multiple arrays into one, and splitting one array into many

**Ufuncs:**

Vectorized operations in NumPy are implemented via ufuncs, whose main purpose is to quickly execute repeated operations on values in NumPy arrays. Ufuncs are extremely flexible.

| **Operator** | **Equivalent ufunc** | **Description** |
| --- | --- | --- |
| + | np.add | Addition (e.g., 1 + 1 = 2) |
| - | np.subtract | Subtraction (e.g., 3 - 2 = 1) |
| - | np.negative | Unary negation (e.g., -2) |
| \* | np.multiply | Multiplication (e.g., 2 \* 3 = 6) |
| / | np.divide | Division (e.g., 3 / 2 = 1.5) |
| // | np.floor\_divide | Floor division (e.g., 3 // 2 = 1) |
| \*\* | np.power | Exponentiation (e.g., 2 \*\* 3 = 8) |
| % | np.mod | Modulus/remainder (e.g., 9 % 4 = 1) |

**Aggregating Func. In Numpy:**

| **Function Name** | **NaN-safe Version** | **Description** |
| --- | --- | --- |
| np.sum | np.nansum | Compute sum of elements |
| np.prod | np.nanprod | Compute product of elements |
| np.mean | np.nanmean | Compute mean of elements |
| np.std | np.nanstd | Compute standard deviation |
| np.var | np.nanvar | Compute variance |
| np.min | np.nanmin | Find minimum value |
| np.max | np.nanmax | Find maximum value |
| np.argmin | np.nanargmin | Find index of minimum value |
| np.argmax | np.nanargmax | Find index of maximum value |
| np.median | np.nanmedian | Compute median of elements |
| np.percentile | np.nanpercentile | Compute rank-based statistics of elements |
| np.any | N/A | Evaluate whether any elements are true |
| np.all | N/A | Evaluate whether all elements are true |

Rules of Broadcasting

Broadcasting in NumPy follows a strict set of rules to determine the interaction between the two arrays:

* Rule 1: If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is *padded* with ones on its leading (left) side.
* Rule 2: If the shape of the two arrays does not match in any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
* Rule 3: If in any dimension the sizes disagree and neither is equal to 1, an error is raised.

**Importing/exporting**

np.loadtxt('file.txt') | From a text file  
np.genfromtxt('file.csv',delimiter=',') | From a CSV file  
np.savetxt('file.txt',arr,delimiter=' ') | Writes to a text file  
np.savetxt('file.csv',arr,delimiter=',') | Writes to a CSV file

**Creating Arrays**

np.array([1,2,3]) | One dimensional array  
np.array([(1,2,3),(4,5,6)]) | Two dimensional array  
np.zeros(3) | 1D array of length 3 all values 0  
np.ones((3,4)) | 3x4 array with all values 1  
np.eye(5) | 5x5 array of 0 with 1 on diagonal (Identity matrix)  
np.linspace(0,100,6) | Array of 6 evenly divided values from 0 to 100  
np.arange(0,10,3) | Array of values from 0 to less than 10 with step 3 (eg [0,3,6,9])  
np.full((2,3),8) | 2x3 array with all values 8  
np.random.rand(4,5) | 4x5 array of random floats between 0–1  
np.random.rand(6,7)\*100 | 6x7 array of random floats between 0–100  
np.random.randint(5,size=(2,3)) | 2x3 array with random ints between 0–4

**Inspecting Properties**

arr.size | Returns number of elements in arr  
arr.shape | Returns dimensions of arr (rows,columns)  
arr.dtype | Returns type of elements in arr  
arr.astype(dtype) | Convert arr elements to type dtype  
arr.tolist() | Convert arr to a Python list  
np.info(np.eye) | View documentation for np.eye

**Copying/sorting/reshaping**

np.copy(arr) | Copies arr to new memory  
arr.view(dtype) | Creates view of arr elements with type dtype  
arr.sort() | Sorts arr  
arr.sort(axis=0) | Sorts specific axis of arr  
two\_d\_arr.flatten() | Flattens 2D array two\_d\_arr to 1D  
arr.T | Transposes arr (rows become columns and vice versa)  
arr.reshape(3,4) | Reshapes arr to 3 rows, 4 columns without changing data  
arr.resize((5,6)) | Changes arr shape to 5x6 and fills new values with 0

**Adding/removing Elements**

np.append(arr,values) | Appends values to end of arr  
np.insert(arr,2,values) | Inserts values into arr before index 2  
np.delete(arr,3,axis=0) | Deletes row on index 3 of arr  
np.delete(arr,4,axis=1) | Deletes column on index 4 of arr

**Combining/splitting**

np.concatenate((arr1,arr2),axis=0) | Adds arr2 as rows to the end of arr1  
np.concatenate((arr1,arr2),axis=1) | Adds arr2 as columns to end of arr1  
np.split(arr,3) | Splits arr into 3 sub-arrays  
np.hsplit(arr,5) | Splits arr horizontally on the 5th index

**Indexing/slicing/subsetting**

arr[5] | Returns the element at index 5  
arr[2,5] | Returns the 2D array element on index [2][5]  
arr[1]=4 | Assigns array element on index 1 the value 4  
arr[1,3]=10 | Assigns array element on index [1][3] the value 10  
arr[0:3] | Returns the elements at indices 0,1,2 (On a 2D array: returns rows 0,1,2)  
arr[0:3,4] | Returns the elements on rows 0,1,2 at column 4  
arr[:2] | Returns the elements at indices 0,1 (On a 2D array: returns rows 0,1)  
arr[:,1] | Returns the elements at index 1 on all rows  
arr<5 | Returns an array with boolean values  
(arr1<3) & (arr2>5) | Returns an array with boolean values  
~arr | Inverts a boolean array  
arr[arr<5] | Returns array elements smaller than 5

**Scalar Math**

np.add(arr,1) | Add 1 to each array element  
np.subtract(arr,2) | Subtract 2 from each array element  
np.multiply(arr,3) | Multiply each array element by 3  
np.divide(arr,4) | Divide each array element by 4 (returns np.nan for division by zero)  
np.power(arr,5) | Raise each array element to the 5th power

**Vector Math**

np.add(arr1,arr2) | Elementwise add arr2 to arr1  
np.subtract(arr1,arr2) | Elementwise subtract arr2 from arr1  
np.multiply(arr1,arr2) | Elementwise multiply arr1 by arr2  
np.divide(arr1,arr2) | Elementwise divide arr1 by arr2  
np.power(arr1,arr2) | Elementwise raise arr1 raised to the power of arr2  
np.array\_equal(arr1,arr2) | Returns True if the arrays have the same elements and shape  
np.sqrt(arr) | Square root of each element in the array  
np.sin(arr) | Sine of each element in the array  
np.log(arr) | Natural log of each element in the array  
np.abs(arr) | Absolute value of each element in the array  
np.ceil(arr) | Rounds up to the nearest int  
np.floor(arr) | Rounds down to the nearest int  
np.round(arr) | Rounds to the nearest int

**Statistics**

np.mean(arr,axis=0) | Returns mean along specific axis  
arr.sum() | Returns sum of arr  
arr.min() | Returns minimum value of arr  
arr.max(axis=0) | Returns maximum value of specific axis  
np.var(arr) | Returns the variance of array  
np.std(arr,axis=1) | Returns the standard deviation of specific axis  
arr.corrcoef() | Returns correlation coefficient of array