1 Data Types

- Before diving into Python for analyzing data, it is necessary to understand some basic concepts about the core Python data types.
- Unlike MATLAB or R, where the default data type has been chosen for numerical work, Python is a general purpose programming language which is very suited to data analysis.
- For example, the basic numeric type in MATLAB is an array (using double precision, which is useful for floating point mathematics), while the basic numeric data type in Python is a 1-dimensional scalar which may be either an integer or a double-precision floating point, depending on the formatting of the number when input.

1.1 Variable Names

Variable names can take many forms, although they can only contain numbers, letters (both upper and lower), and underscores (_).
They must begin with a letter or an underscore and are CaSe SeNsItIve. Additionally, some words are reserved in Python and so cannot be used for variable names (e.g. `import` or `for`). For example,

```
x = 1.0
X = 1.0
X1 = 1.0
X1 = 1.0
x1 = 1.0
dell = 1.0
dellreturns = 1.0
dellReturns = 1.0
_x = 1.0
x_ = 1.0
```

are all legal and distinct variable names. Note that names which begin or end with an underscore, while legal, are not normally used since by convention these convey special meaning. Illegal names do not follow these rules.

```
>>> x = []
>>> type(x)
builtins.list
>>>
>>> x=[1,2,3,4]
>>> x
[1,2,3,4]
# 2-dimensional list (list of lists)
>>> x = [[1,2,3,4], [5,6,7,8]]
>>> x
[[1, 2, 3, 4], [5, 6, 7, 8]]
# Jagged list, not rectangular
>>> x = [[1,2,3,4] , [5,6,7]]
>>> x
[[1, 2, 3, 4], [5, 6, 7]]
>>>
# Mixed data types
>>> x = [1,1.0,1+0j,'one',None,True]
>>> x
[1, 1.0, (1+0j), 'one', None, True]
```

2 Core Native Data Types

2.1 Numeric

- Simple numbers in Python can be either integers, floats or complex. Integers correspond to either 32 bit or 64-bit integers, depending on whether the python interpreter was compiled for a 32-bit or 64-bit operating system, and floats are always 64-bit (corresponding to doubles in C/C++).
- Long integers, on the other hand, do not have a fixed size and so can accommodate numbers which are larger than maximum the basic integer type can handle.
- We will not cover all Python data types, and instead focus on those which are most relevant for data analysis and statistics.

2.1.1 Floating Point (float)

The most important (scalar) data type for numerical analysis is the float. Unfortunately, not all noncomplex numeric data types are floats. To input a floating data type, it is necessary to include a "." (full-stop /period) in the expression. This example uses the function `type()` to determine the data type of a variable.

```
>>> x = 1
>>> type(x)
int
>>> x = 1.0
>>> type(x)
float
>>> x = float(1)
>>> type(x)
float
```

This example shows that using the expression that `x = 1` produces an integer-valued variable while `x = 1.0` produces a float-valued variable. Using integers can produce unexpected results and so it is important to include "`.0`" when expecting a float.

2.1.2 Complex (complex)

Complex numbers are alsooften very important for scientific computing. Complex numbers are created in Python using `j` or the function `complex()`.

```
>>> x = 1.0
>>> type(x)
float
>>> x = 1j
>>> type(x)
complex
>>> x = 2 + 3j
>>> x
(2+3j)
>>> x = complex(1)
>>> x
(1+0j)
```

Note that `a+bj`is the same as `complex(a,b )`, while `complex(a)` is the same as `a+0j`.

2.1.3 Integers (`int` and `long`)

- Floats use an approximation to represent numbers which may contain a decimal portion. The integer data type stores numbers using an exact representation, so that no approximation is needed.
- The cost of the exact representation is that the integer data type cannot express anything that isn't an integer, rendering integers of limited use in most numerical work.
- Basic integers can be entered either by excluding the decimal, or explicitly using the `int()` function.
- The `int()` function can also be used to convert a float to an integer by round towards 0.

]

```
>>> x = 1
>>> type(x)
int
```

```
>>> x = 1.0
>>> type(x)
float
>>> x = int(x)
>>> type(x)
int
```

Integers can range from -2^{31} to $2^{31}-1$. Python contains another type of integer, known as a long integer, which has no effective range limitation. Long integers are entered using the syntax `x = 1L` or by calling `long()`. Additionally python will automatically convert integers outside of the standard integer range to long integers.

```
>>> x = 1
>>> x
1
>>> type(x)
int
>>> x
1L
>>> type(x)
long
>>> x = long(2)
>>> type(x)
long
>>> y = 2
>>> type(y)
int
>>> x = y ** 64 # ** is denotes exponentiation, y^64 in TeX
>>> x
18446744073709551616L
```

2.1.4 Boolean (`bool`)

- The Boolean data type is used to represent true and false, using the reserved keywords `True` and `False`.
- Boolean variables are important for program flow control and are typically created as a result of logical operations , although they can be entered directly.

```
>>> x = True
>>> type(x)
bool
>>> x = bool(1)
>>> x
True
>>> x = bool(0)
>>> x
False
```

Non-zero, non-empty values generally evaluate to true when evaluated by `bool()`. Zero or empty values such as `bool(0), bool(0.0), bool(0.0j), bool(None), bool('')` and `bool([])` are all false.

2.1.5 Strings (`str`)

Strings are not usually important for numerical analysis, although they are frequently encountered when dealing with data files, especially when importing or when formatting output for human consumption. Strings are delimited using ' or "" but not using combination of the two delimiters (i.e. do not try "') in a single string, except when used to express a quotation.

```
>>> x = 'abc'
>>> type(x)
34
str
>>> y = '"A quotation!"'
>>> print(y)
"A quotation!"
```

2.1.6 Lists (`list`)

- Lists are a built-in data type which require other data types to be useful.
- A list is a collection of other objects – floats, integers, complex numbers, strings or even other lists.
- Lists are essential to Python programming and are used to store collections of other values. For example, a list of floats can be used to express a vector (although the NumPy data types `array` and `matrix` are better suited).
- Lists also support slicing to retrieve one or more elements.
- Basic lists are constructed using square braces, [], and values are separated using commas.

```
>>> x = []
>>> type(x)
builtins.list
>>> x=[1,2,3,4]
>>> x
[1,2,3,4]
# 2dimensional
list (list of lists)
>>> x = [[1,2,3,4], [5,6,7,8]]
>>> x
[[1, 2, 3, 4], [5, 6, 7, 8]]
# Jagged list, not rectangular
>>> x = [[1,2,3,4] , [5,6,7]]
>>> x
[[1, 2, 3, 4], [5, 6, 7]]
# Mixed data types
>>> x = [1,1.0,1+0j,'one',None,True]
>>> x
[1, 1.0, (1+0j), 'one', None, True]
```

These examples show that lists can be regular, nested and can contain any mix of data types including other lists.

2.1.7 Xrange (xrange)

- A xrange is a useful data type which is most commonly encountered when using a `for` loop.
- `xrange(a,b,i)` creates the sequences that follows the pattern $a, a+i, a+2i, ..., a+(m-1)i$ where m is the stepsize.
- In other words, it find all integers x starting with a such $a \leq x < b$ and where two consecutive values are separated by i .
- xrange can be called with 1 or two parameters – xrange(a,b) is the same as xrange(a,b,1) and `xrange(b)` is the same as `xrange(0,b,1)`.

```
>>> x = xrange(10)
>>> type(x)
xrange
>>> print(x)
xrange(0, 10)
>>> list(x)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> x = xrange(3,10)
>>> list(x)
[3, 4, 5, 6, 7, 8, 9]
>>> x = xrange(3,10,3)
>>> list(x)
[3, 6, 9]
>>> y = range(10)
>>> type(y)
list
>>> y
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- xrange is not technically a list, which is why the statement `print(x)` returns `xrange(0,10)`.
- Explicitly converting with list produces a list which allows the values to be printed. Technically xrange is an iterator which does not actually require the storage space of a list.
- This can be seen in the differences between using `y = range(10)`, which returns a list and `y=xrange(10)` which returns an xrange object.
- Best practice is to use `xrange` instead of `range`.