Python Tutorial – Part I: Introduction

Mark A. Austin

University of Maryland

austin@umd.edu
ENCE 688P, Spring Semester 2022

January 18, 2023

Overview

Part 2

Data Types

(Data Types in Python)

Builtin Data Types

```
dtype              Description
==============================================================
Text Type:         str
Numeric Types:     int, float, complex
Sequence Types:    list, tuple, range
Mapping Type:      dict
Set Types:         set, frozenset
Boolean Type:      bool
Binary Types:      bytes, bytearray, memoryview
None Type:         NoneType
==============================================================
```

Example 1: Getting an int data type ...

```
a = 1
print ( type(a) )
```

Output:

```
< class 'int' >
```

Builtin Data Types

Example 2: Float, complex, boolean, string and list types ...

```
b = 1.5                  # <-- define float ...
print ( type(b) )
c = 1.0 + 1.5j           # <-- define complex ...
print ( type(c) )
d = True                 # <-- define boolean ...
print ( type(d) )
e = "this is a string"   # <-- define string ...
print ( type(e) )
f = ["A, "B", "C", "D"]  # <-- define list ...
print ( type(f) )
```

Output:

```
< class 'float' >
< class 'complex' >
< class 'bool' >
< class 'str' >
< class 'list' >
```

Builtin Data Types

Example 3: Formatting data type output ...

```
print("--- a = {:2d} ... ".format(a) );      # <-- Format integer output.
print("--- b = {:.2f} ... ".format(b) );     # <-- two-decimal places
print('--- c = {:.2f}'.format(c))            #     of accuracy.
print("--- d = {:.5s} ... ".format( str(d) ))
print("--- e = {:15s} ... ".format(e) )
output = ["%.5s" % elem for elem in f ]      # <-- convert list to string ...
print("--- f = ", output )
```

Output:

```
--- a =  1 ...
--- b = 1.50 ...
--- c = 1.00+1.50j
--- d = True ...
--- e = this is a string ...
--- f =  ['A', 'B', 'C', 'D']
```

Floating-Point Numbers

Definition. Floating point variables and constants are used represent values outside of the integer range (e.g., `3.4`, `-45.33` and `2.714`) and are either very large or small in magnitude, (e.g., `3.0e-25`, `4.5e+05`, and `2.34567890098e+19`).

IEEE 754 Floating-Point Standard. Specifies that a floating point number take the form:

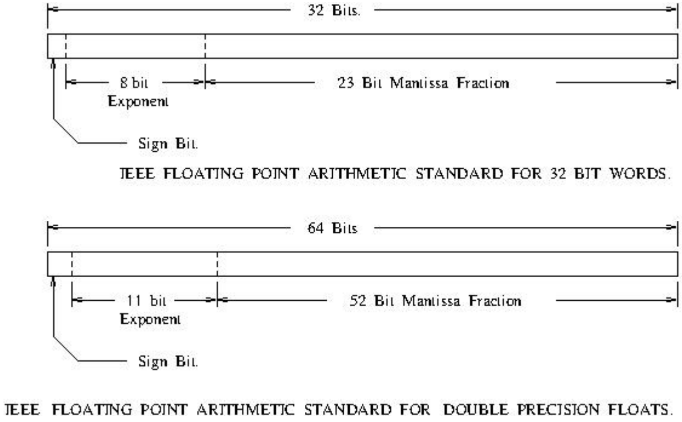
$$X = \sigma \cdot m \cdot 2^E. \quad (1)$$

Here:

- σ represents the sign of the number.
- m is the mantissa (interpreted as a fraction $0 < m < 1$).
- E is the exponent.

IEEE 754 Floating-Point Standard

Ensures floating point implementions and arithmetic are consistent across various types of computers (e.g., PC and Mac).

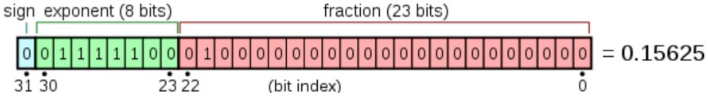


Largest and Smallest Floating-Point Numbers

```
====================================================================
                  Default
Type   Contains     Value   Size      Range and Precision
====================================================================

float  IEEE 754       0.0   32 bits   +- 13.40282347E+38 /
       floating point                 +- 11.40239846E-45

       Floating point numbers are represented to approximately
       6 to 7 decimal places of accuracy.

double IEEE 754       0.0   64 bits   +- 11.7976931348623157OE+308 /
       floating point                 +- 14.94065645841246544E-324

       Double precision numbers are represented to approximately
       15 to 16 decimal places of accuracy.
====================================================================
```

Working with Double Precision Numbers

Simple Example. Here is the floating point representation for 0.15625



Note. Keep in mind that floating-point numbers are stored in a binary format – this can lead to surprises.

For example, when the decimal fraction 1/10 (0.10 in base 10) is converted to binary, the result is an expansion of infinte length.

Bottom line: You cannot store 0.10 precisely in a computer.

Working with Double Precision Numbers

Accessing the Math Library Module

```
import math;  # <-- import the math library ...
```

Math Constants

```
Method          Description
=========================================================================
math.e          Returns Euler's number (2.7182 ...).

math.inf        Returns floating-point positive infinity.

math.pi         Returns PI (3.1415926 ...).
=========================================================================
```

Math Methods

```
Method          Description
=========================================================================
math.acos()     Returns the arc cosine of a number.
math.acosh()    Returns the inverse hyperbolic cosine of a number.
math.asin()     Returns the arc sine of a number.
math.asinh()    Returns the inverse hyperbolic sine of a number.
=========================================================================
```

Working with Double Precision Numbers

Math Methods (continued) ...

```
Method              Description
====================================================================
math.atan()         Returns the arc tangent of a number in radians
math.atan2()        Returns the arc tangent of y/x in radians
math.ceil()         Rounds a number up to the nearest integer
math.cos()          Returns the cosine of a number
math.cosh()         Returns the hyperbolic cosine of a number
math.exp()          Returns E raised to the power of x
math.fabs()         Returns the absolute value of a number
math.floor()        Rounds a number down to the nearest integer
math.gcd()          Returns the greatest common divisor of two integers
math.isfinite()     Checks whether a number is finite or not
math.isinf()        Checks whether a number is infinite or not
math.isnan()        Checks whether a value is NaN (not a number) or not
math.isqrt()        Rounds a square root number down to the nearest integer
math.ldexp()        Returns the inverse of math.frexp() which is
                    x * (2**i) of the given numbers x and i
math.lgamma()       Returns the log gamma value of x
====================================================================
```

Working with Double Precision Numbers

Math Methods (continued) ...

```
Method             Description
====================================================================
math.log()         Returns the natural logarithm of a number, or the
                   logarithm of number to base.
math.log10()       Returns the base-10 logarithm of x
math.log1p()       Returns the natural logarithm of 1+x
math.log2()        Returns the base-2 logarithm of x
math.perm()        Returns the number of ways to choose k items from n
                   items with order and without repetition
math.pow()         Returns the value of x to the power of y
math.prod()        Returns the product of all the elements in an iterable
math.radians()     Converts a degree value into radians
math.remainder()Returns the closest value that can make numerator
                   completely divisible by the denominator
math.sin()         Returns the sine of a number
math.sinh()        Returns the hyperbolic sine of a number
math.sqrt()        Returns the square root of a number
math.tan()         Returns the tangent of a number
math.tanh()        Returns the hyperbolic tangent of a number
math.trunc()       Returns the truncated integer parts of a number
====================================================================
```

Working with Double Precision Numbers

Example 4: Formatting PI ...

```
import math;      # <-- import math library.
PI = math.pi;     # <-- create user-defined constant.

print("--- PI = {:.2f} ...".format(PI) );  # <-- 2 decimal places.
print("--- PI = {:.15f} ...".format(PI) ); # <-- 15 decimal places.
print("--- PI = {:8.2f} ...".format(PI) ); # <-- 8 characters wide,
                                           #     2 decimal places.
print("--- PI = {:16.12f} ...".format(PI) );# <-- 16 characters wide,
                                            #     12 decimal places.
print("--- PI = {:16.6e} ...".format(PI) ); # <-- exponential format.
```

Output:

```
--- PI = 3.14 ...
--- PI = 3.141592653589793 ...
--- PI =     3.14 ...
--- PI =   3.141592653590 ...
--- PI =     3.141593e+00 ...
```

First Program

(Evaluate and Plot Sigmoid Function)

Problem Desription

Problem Description

In neural network models, the sigmoid function:

$$\sigma(x) = \left[\frac{1}{1 + e^{-x}} \right]. \quad (2)$$

serves as an activation. Our first program evaluates and plots $\sigma(x)$ over the range $x \in [-10, 10]$.

Running the Program

From the terminal window, simply type:

```
prompt >> python3 TestSigmoidFunction.py
```

Evaluate and Plot Sigmoid Function

The Python interpreter/compiler will complain if one or more of the required packages (e.g., matplotlib) are missing.

Use pip to install missing Python Packages

The standard package-management system used to install and manage software packages is called pip (or pip3).

Example: And installation is easy!

```
prompt >> pip3 install numpy
prompt >> pip3 install matplotlib
```

To get a list of installed packages:

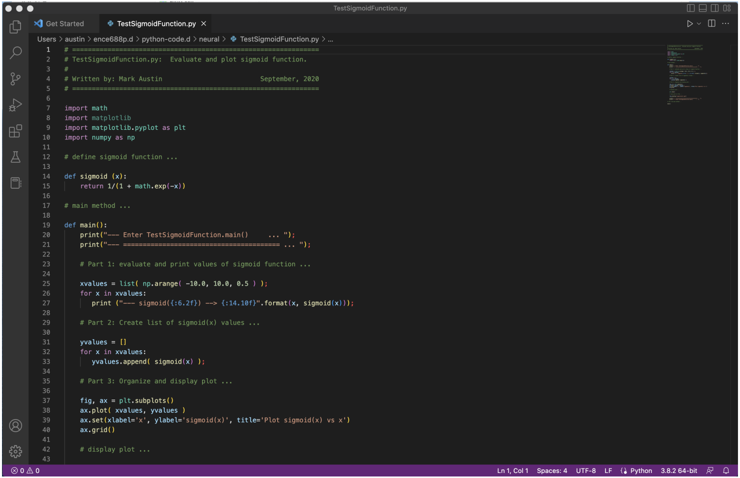
```
prompt >> pip3 list
```

Evaluate and Plot Sigmoid Function

Abbreviated Output:

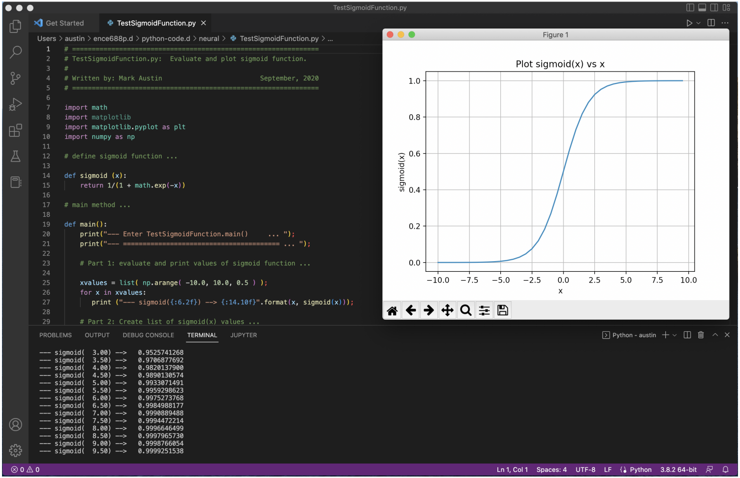
```
Package                Version
---------------------- ---------
....
jupyter                1.0.0
Keras                  2.4.3
....
matplotlib             3.4.1
....
numpy                  1.19.5
....
pandas                 1.1.5
....
scikit-learn           0.24.2
scipy                  1.6.2
....
sklearn                0.0
```

Program Source Code in Visual Studio Code




```python
# ================================================================
# TestSigmoidFunction.py:  Evaluate and plot sigmoid function.
#
# Written by: Mark Austin                         September, 2020
# ================================================================

import math
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

# define sigmoid function ...

def sigmoid (x):
    return 1/(1 + math.exp(-x))

# main method ...

def main():
    print("--- Enter TestSigmoidFunction.main()     ... ");
    print("--- ======================================== ... ");

    # Part 1: evaluate and print values of sigmoid function ...

    xvalues = list( np.arange( -10.0, 10.0, 0.5 ) );
    for x in xvalues:
        print ("--- sigmoid({:6.2f}) --> {:14.10f}".format(x, sigmoid(x)));

    # Part 2: Create list of sigmoid(x) values ...

    yvalues = []
    for x in xvalues:
        yvalues.append( sigmoid(x) );

    # Part 3: Organize and display plot ...

    fig, ax = plt.subplots()
    ax.plot( xvalues, yvalues )
    ax.set(xlabel='x', ylabel='sigmoid(x)', title='Plot sigmoid(x) vs x')
    ax.grid()

    # display plot ...
```

Program Source Code + Output in Visual Studio Code



Program Source Code

```
# ============================================================
# TestSigmoidFunction.py: Evaluate/plot sigmoid function.
#
# Written by: Mark Austin                      September, 2020
# ============================================================

import math
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

# define sigmoid function ...

def sigmoid (x):
    return 1/(1 + math.exp(-x))

# main method ...

def main():
    print("--- Enter TestSigmoidFunction.main() ...");
    print("--- ================================ ...");

    # Part 1: Evaluate and print sigmoid function

    xvalues = list( np.arange( -10.0, 10.0, 0.5 ) );
    for x in xvalues:
       print ("--- sigmoid({:6.2f}) --> {:14.10f}".format(x, sigmoid(x)));

    # Part 2: Create list of sigmoid(x) values ...
```

Program Source Code

```
    # Part 2: Create list of sigmoid(x) values ...

    yvalues = []
    for x in xvalues:
       yvalues.append( sigmoid(x) );

    # Part 3: Organize and display plot ...

    fig, ax = plt.subplots()
    ax.plot( xvalues, yvalues )
    ax.set(xlabel='x', ylabel='sigmoid(x)',
           title='Plot sigmoid(x) vs x')
    ax.grid()

    # display and save plot ...

    plt.show()

    fig.savefig("sigmoid-plot.jpg")

    print("--- ================================ ...");
    print("--- Leave TestSigmoidFunction.main() ...");

# call the main method ...

main()
```

Program Source Code

Points to Note:

- Line comment statements begin with the # character.
- Lines 7-10 import the math, matplotlib, matplotlib.pyplot and numpy modules to the program, and make the functions therein available.
- Functions are the primary method of code organization and reuse in Python.
- User-defined functions are declared with the `def` keyword. A function contains a block of code with an optional `return` keyword.
- Lines 13-14 evaluate and return the sigmoid function.
- Use of the second function, `main()`, is a carry over from programming with C, where all programs begin their execution in `main()`. Its use in Python is optional.

Program Source Code

Points to Note (continued):

- Line 25 creates a list of x coordinates. The numpy function `np.arange()` covers $[-10, 10]$ in increments of 0.5.
- Lines 26-27 systematically traverse the elements of `xvalues`, and compute and print the corresponding values of the `sigmoid()` function.
- Line 27 formats and prints the output. The specification `{:6.2}f` means that the output should be printed as a floating point number, six characters wide, and with two decimal places of accuracy to the right of the decimal point.
- Lines 31-33 traverse the elements of `xvalues`, and systematically assemble a list of sigmoid function `yvalues`.
- Lines 37-47 format a plot of `yvalues` vs `xvalues`, and save to `sigmoid-plot.jpg`.