Python for MATLAB® Users

Jian Tao

jtao@tamu.edu

Fall 2017 HPRC Short Course

10/19/2017

Relevant Short Courses and Workshops

Introduction to MATLAB Programming

https://hprc.tamu.edu/training/intro_matlab.html

Introduction to Python

https://hprc.tamu.edu/training/intro_python.html

Introduction to Scientific Python

https://hprc.tamu.edu/training/intro_scientific_python.html

Bring-Your-Own-Code Workshop

https://coehpc.engr.tamu.edu/byoc/

Offered regularly

MATLAB

MATLAB is a commercial **software tool** (language + IDE) for:

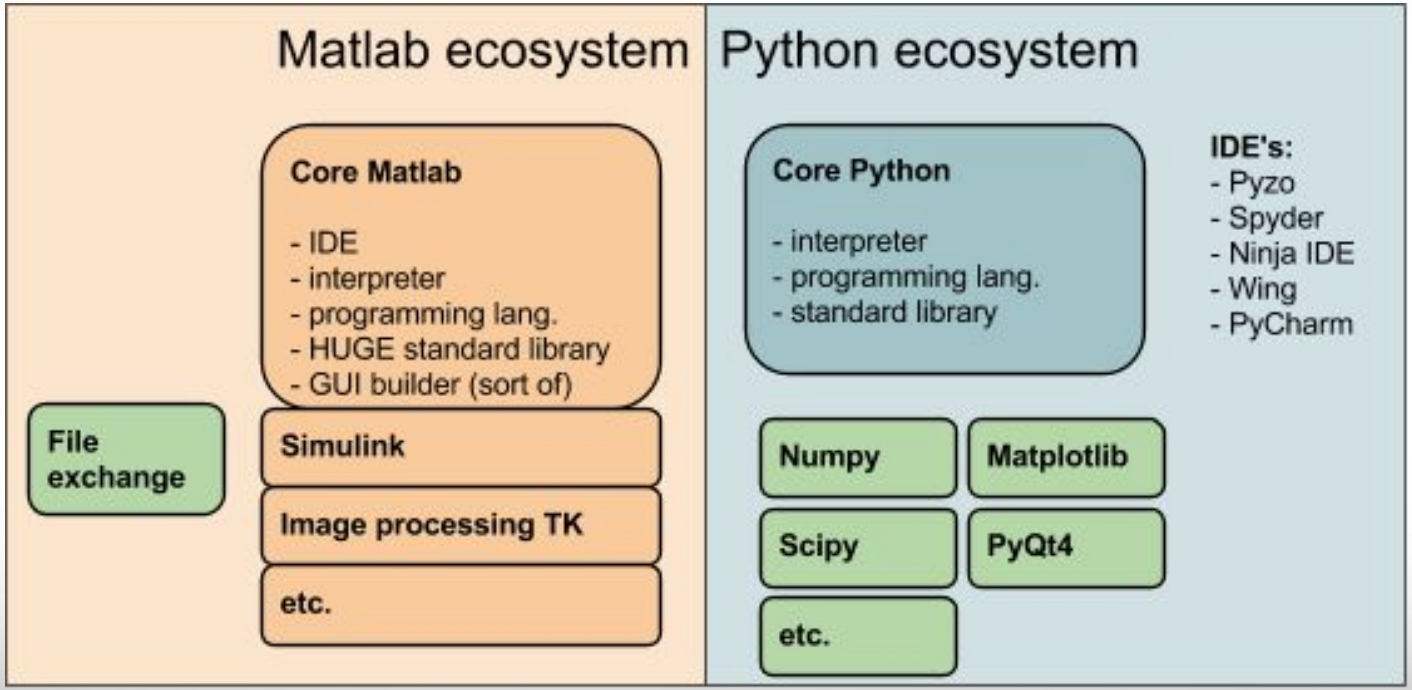
- Performing mathematical computations and signal processing
- Analyzing and visualizing data (excellent graphics tools)
- Modeling physical systems and phenomena
- Testing engineering designs
- Functionalities enhanced via apps & packages
- https://www.mathworks.com/help/matlab/index.html

Python

Python is an open source **programming language** that is empowered by a large selection of open source libraries:

- With NumPy, one can operate large, multi-dimensional arrays and matrices.
- With Matplotlib, one can visualize various types of data.
- With TensorFlow, one can build applications with machine intelligence.
- With Django, one can build sophisticated web sites.
- and many more…
- https://www.python.org/about/gettingstarted/

Round One: Python vs MATLAB



http://www.pyzo.org

Round Two: Python vs MATLAB

Python

Free & open source for both core & extra libraries.

Relatively small community in science & engineering, but rapidly growing.

Installation of different packages to set up a functional development environment which is not on par with that of MATLAB so far.

MATLAB

Expensive core & extra packages & proprietary algorithms.

Large scientific community & legacy code base.

Single installation with a user friendly Integrated Development Environment & wonderful plotting tools.

http://www.pyzo.org

Python as an Advanced Calculator

Arithmetic Operators

```
+     Addition
-     Subtraction
/     division
%     mod
*     multiplication
//    integer division
**    to the power of
```

Arithmetic Operators and Order of Operations

- Addition (+), Subtraction (-), Multiplication (*), Division (/), Power (**)
- Order of Operations (same rules you should already know from math class and using a calculator)

    1. Complete all calculations inside parentheses or brackets using the precedent rules below
    2. Powers (left to right)
    3. Multiplication and Division (left to right)
    4. Addition and Subtraction (left to right)

Arithmetic Expressions

Some examples:

```
>>>10/5*2
>>>5*2**3+4*2   #^ -> **
>>>-1**4
>>>8**1/3
>>>pi           #pi is not defined
```

Relational Operators

```
==      True, if it is equal
!=      True, if not equal to
<       less than
>       greater than
<=      less than or equal to
>=      greater than or equal to
```

Boolean and Bitwise Operators

`and`	Logical and
`or`	Logical or
`not`	Not
`^`	Exclusive OR
`|`	Bitwise OR
`~`	Negate
`&`	Bitwise And
`>>`	Right shift
`<<`	Left shift

Variables

The basic types of Python include **float**, **int**, **str**, **complex**, and **bool**. A variable can be deleted with **del**

```
>>>b = True
>>>whos              # works on iPython!
>>>type(b)           # type of the variable
>>>x = "Hi"
>>>y = 10
>>>z = complex(1, 2)
>>>print(b, x, y, z)
>>>del b; print(b)
```

Naming Rules for Variables

- Variable names must begin with a letter

```
>>>4c = 12
```

- Names can include any combinations of letters, numbers, and underscores

```
>>>c_4 = 12
```

- Maximum length for a variable name is not limited
- Python is case sensitive. The variable name A is different than the variable name a.

Exercise

Create two variables: `a = 4` and `b = 17.2`

Now use Python to perform the following set of calculations:

$(b+5.4)^{1/3}$ $\qquad$ $b^2-4b+5a$

`a>b and a>1.0` $\qquad$ `a!=b`

Basic Syntax for Statements

1. No spaces or tab are allowed at the start of a statement.
2. Comments start with '**#**'
3. Statements finish at the end of the line ('****' can e used to indicate an unfinished line)

```
>>>1+2
>>> name = 'Adam'        # Indentation Error!
>>>months = "Aug\
…"
```

Displaying Variables

We can display a variable (i.e., show its value) by simply typing the name of the variable at the command prompt (leaving off the semicolon).

We can also use **`print`** to display variables.
Type the following commands at the command prompt:

```
>>>print('The value of x is:'); print(x)
```

Numerical Data Types - NumPy

Name	Range
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 (real and imaginary components)
complex128	Complex number, represented by two 64-bit floats (real and imaginary components)

Why should I care how data is stored in a computer?

Perform each of the following calculations in your head.

```
a = 4/3
b = a - 1
c = 3*b
e = 1 - c
```

What does Python get?

Why should I care how data is stored in a computer?

What does Python get?

```
>>>a = 4/3    #1.3333333333333333
>>>b = a - 1  #0.3333333333333326
>>>c = 3*b    #0.9999999999999998
>>>e = 1 - c  #2.220446049250313e-16
```

It is not possible to perfectly represent all real numbers using a finite string of 1s and 0s.

Creating Strings (Text Variables)

Single quotes & double quotes are the same in Python. '\' can be used if there is a quote in a string.

```
>>>month = 'Aug'
>>>my_str1 = 'Tom\'s toy'
>>>my_str2 = "Tom's toy"
```

Some Useful Math Functions

Function	Python	Function	Python
cosine	cos	square root	sqrt
sine	sin	exponential	exp
tangent	tan	logarithm (base 10)	log10
arc cosine	acos	natural log (base e)	log
arc sine	asin	round to nearest integer	round
arc tangent	atan	round down to integer	floor
Euclidean norm	hypot	round up to integer	ceil

Note: All the functions are in the math package, which must be imported before these functions can be used. Trigonometric functions assume input in radians. Euclidean norm returns `sqrt(x*x + y*y)`.

Help

- The **`help`** command provide information about a function. Type **`help("math.cos")`** at the command prompt. This only works if you know the name of the function you want help with.
- Type **`help()`** to get the interactive help utility

```
help>math.cos
```

Data Structures: Tuple, List, Set, & Dictionary

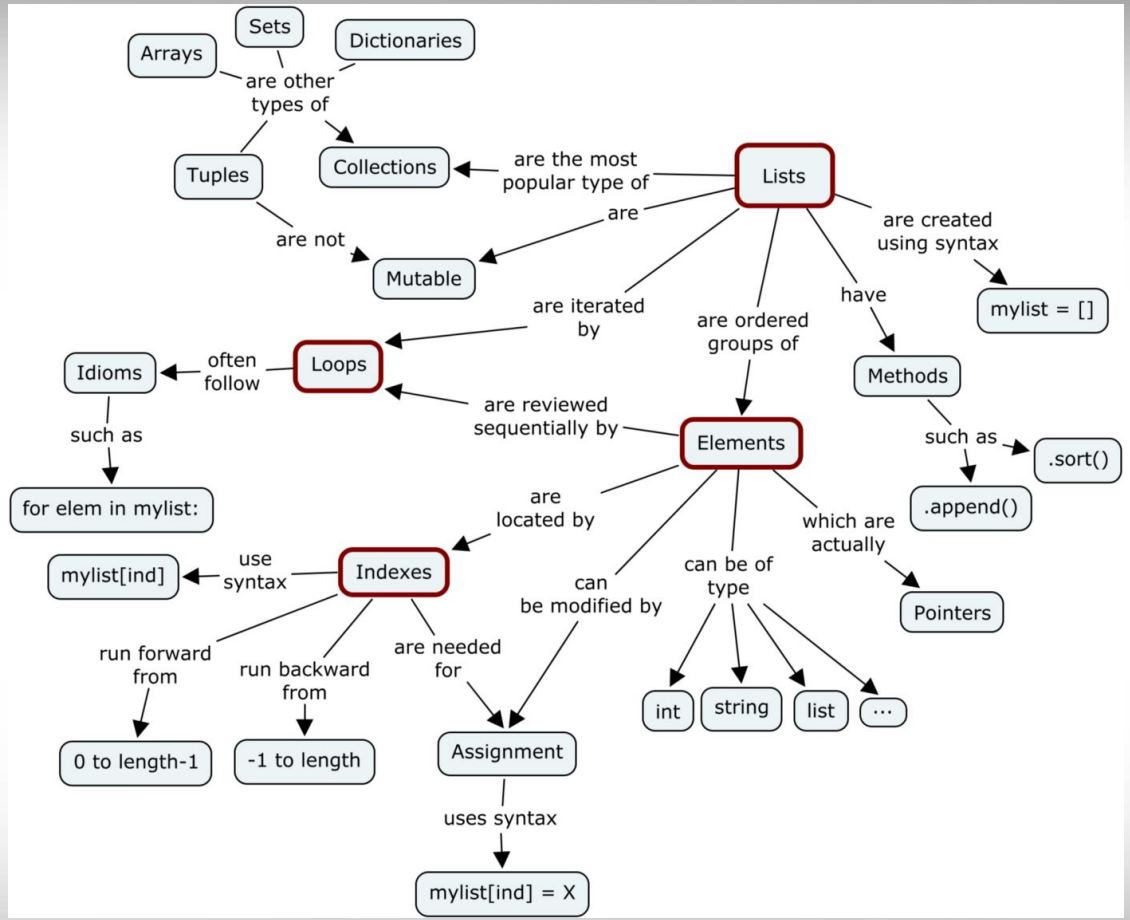


Image Credit: https://medium.com/@meghamohan

Lists & Tuples

- **Lists** are the most commonly used data structure.
- A sequence of data that is enclosed in square brackets.
- Each element can be accessed by calling its index starting with 0.
- Lists are declared with `list()` or `[]`.

- **Tuples** are similar to lists.
- The elements in a list can be changed (mutable), but in tuple they can not be changed (immutable).
- Each element can be accessed by calling its index starting with 0.
- Tuples are declared with `tuple()` or `()`.

Lists & Tuples

List examples:

```
>>>months=['Oct','Nov','Dec']
>>>names=list()
>>>type(months)
>>>months[0]
>>>months[-1]      #try others
>>>months[0] = 'Sep'
>>>months
>>>m=tuple(months)
```

Tuple examples:

```
>>>months=('Oct','Nov','Dec')
>>>names=tuple()
>>>type(months)
>>>months[0]
>>>months[-1]      #try others
>>>months[0] = 'Sep'
>>>months
>>>m=list(months)
```

Sets

- Sets are mainly used to eliminate repeated numbers in a sequence/list.
- It is also used to perform some standard set operations.
- Sets are declared with `set()` or `{}`
- Also `set([sequence])` can be executed to declare a set with elements

Examples:

```
>>>months={'Oct','Nov','Dec','Dec'}
>>>names=set()
>>>type(months)
>>>months[0]          #error!
>>>months.add('Sep')
>>>m=list(months)
```

Dictionaries

- **Dictionaries** are mappings between keys and items stored in the dictionaries.
- Alternatively one can think of dictionaries as sets in which something stored against every element of the set.
- Dictionaries are mutable.
- To define a dictionary, equate a variable to `{}` or `dict()`

Examples:

```
>>>months={'Oct':'October','Nov':'Nov
ember','Dec':'December'}
>>>names=dict()
>>>type(months)
>>>months['Oct']
>>>months.update({'Sep':'September'})
>>>months['Oct']='Current Month'
>>>m=list(months)
```

Conditional Statements & Loops

Loop Control Statements - *for*

for statements help repeatedly execute a block of code for a certain number of iterations

```
%matlab
x = zeros(1,10);
for n = 1 : 10
    x(n) = n;
end
```

```
#python
import numpy as np
x = np.zeros(10);
for n in range(10):
    x[n] = n
```

Loop Control Statements - *while*

while statements repeatedly execute a block of code as long as a condition is satisfied.

```
%matlab
n = 1;
sum = 0;
while n <= 100
    sum = sum + n;
    n = n + 1;
end
```

```
#python
n = 1
sum = 0
while n <= 100:
    sum = sum + n
    n = n + 1
```

Conditional Statements

Execute statements if condition is true

```
%matlab
if a>10
    disp('a > 10');
elseif a<10
    disp('a < 10')
else
    disp('a = 10')
end
```

```
#python
if a>10:
    print('a > 10');
elif a<10:
    print('a < 10')
else:
    print('a = 10')
```

Nested-Loop: Simple Example

```
%matlab
for r = 1:4
  for c = 1:4
    fprintf('(%i,%i)\n',r,c);
  end
end
```

```
#python
for r in range(1,5):
  for c in range(1,5):
    print('(%i,%i)'%(r, c))
```

Adding Break Statements

What if we add a break statement in the outer loop?

```
%matlab
for r = 1:4
  if r == 2
    break;
  end
  for c = 1:4
    fprintf('(%i,%i)\n',r,c);
  end
end
```

```
#python
for r in range(1,5):
  if r == 2:
    break
  for c in range(1,5):
    print('(%i,%i)'%(r, c))
```

Adding Break Statements

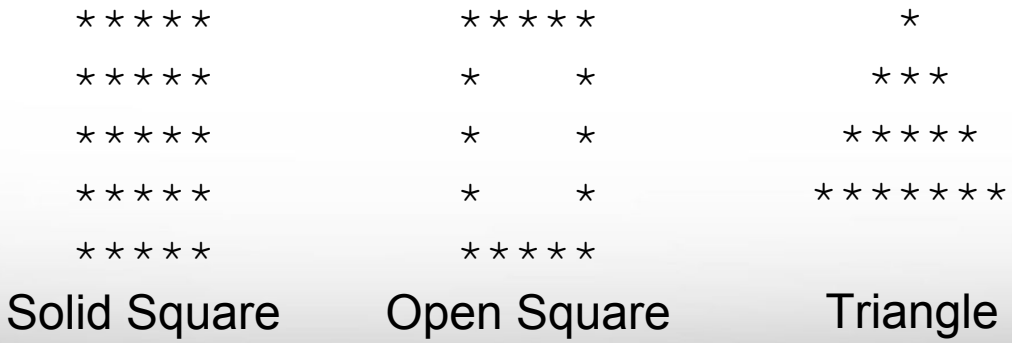
What if we add a break statement in the inner loop?

```
%matlab
for r = 1:4
  end
  for c = 1:4
    if r == 2
      break;
    fprintf('(%i,%i)\n',r,c);
  end
end
```

```
#python
for r in range(1,5):
  for c in range(1,5):
    if r == 2:
      break
    print('(%i,%i)'%(r, c))
```

Exercise: Output

Write a script that will display each of the following shapes using asterisks *



Solid Square Open Square Triangle

Function

```
%Matlab
function[out]=myabs(number)
            if   number   >   0
                 out   =   number
                           else
                out   =   -number
                           end
end
```

```
#Python
def myabs(number):
          if   number   >   0:
               return  number
                           else:
          return -number
```

NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Create an Array

There are multiple ways to create an array with numpy.

```
>> %matlab
>> a = [1 2 3 4 5 6 7 8 9]
>> b = [1 2 3;4 9 6;7 8 9]
>> c = zeros(3,3)
>> d = ones(3,3)
>> e = magic(8)
>> f = 0:10:100
>> g = rand(3,5)
>> h = eye(5)
```

```
>>>#python
>>>import numpy as np
>>>a = np.array([1,2,3,4,5,6,7,8,9])
>>>b = np.array([[1,2,3],[4,9,6],[7,8,9]])
>>>c = np.zeros((3,3))
>>>d = np.ones((3,3))
>>>#no function to create magic matrix
>>>f = np.arange(0,100,10)
>>>g = np.random.uniform(0,1,(3,5))
>>>h = np.eye(5)
```

Array & Matrix Operations - II

```
>> %matlab
>> b + 10
>> sin(b)
>> b'
>> inv(b)
>> b*inv(b)
>> b.*b
>> b.^2
```

```
>>>#python
>>>b + 10                #each element + 10
>>>np.sin(b)             #sin function
>>>b.transpose()         #transpose
>>>c=np.linalg.inv(b)    #inverse
>>>np.dot(b,c)           #matrix multiplication
>>>b*c                   #element-wise multiplication
>>>b**2                  #element-wise square
```

Array & Matrix Operations - III

Numpy arrays can be concatenated

```
>> %matla
>> B_H = [b, b]   %horizontal
>> B_V = [b; b]   %vertical
```

```
>>>#python
>> B_H = np.concatenate((b, b), axis=1)
>> B_V = np.concatenate((b, b), axis=0)
```

Python index goes starting from 0 while MATLAB index starting from 1.

```
>> %matlab
>> g(3, 5)
>> g(1:3, 5)
>> g(3, :)
```

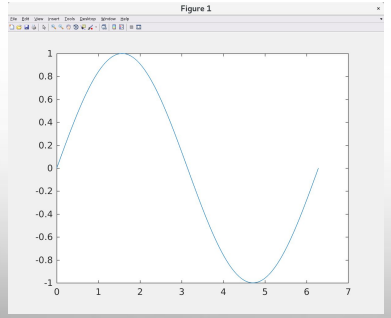
```
>> %pyhton
>> g[2, 4]
>> g[0:2, 4]
>> g[2, :]
```

Plots with Matplotlib

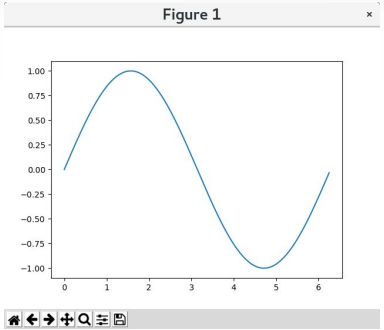
Simple Line Plot

Matplotlib is a widely used Python plotting library.

```
>>%matlab
>>x = 0:pi/100:2*pi;
>>y = sin(x);
>>plot(x,y)
```



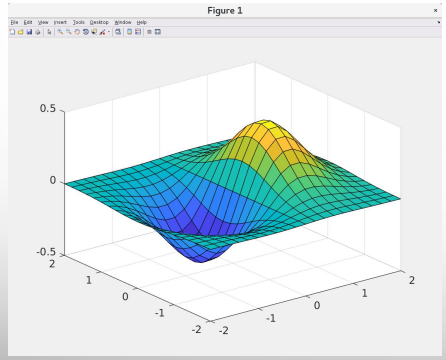

```
>>>#python - matplotlib
>>>import numpy as np
>>>import matplotlib.pyplot as plt
>>>x = np.arange(0,np.pi*2,
np.pi/100)
>>>y = np.sin(x)
>>>plt.plot(x, y)
>>>plt.show()
```



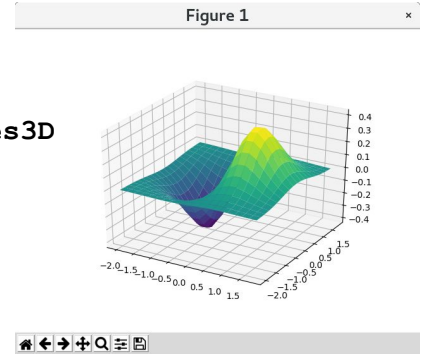
Simple Surface Plot

Surface plot typically display a surface defined by a function in two variables, **z=f(x,y)**.

```
>>%matlab
>>[x,y] = meshgrid(-2:.2:2);
>>z = x .* exp(-x.^2 - y.^2);
>>figure %new figure window
>>surf(x,y,z)
```




```
>>>#python
>>>from mpl_toolkits.mplot3d import Axes3D
>>>import matplotlib.pyplot as plt
>>>from matplotlib import cm
>>>import numpy as np
>>>fig = plt.figure()
>>>ax = fig.gca(projection='3d')
>>>x, y = np.meshgrid(np.arange(-2, 2, 0.2), \
   np.arange(-2, 2, 0.2))
>>>z = x*np.exp(-(x**2 + y**2))
>>>surf = ax.plot_surface(x, y, z, cmap=cm.viridis)
>>>plt.show()
```



Online Resources

Official Python Tutorial

https://docs.python.org/3.6/tutorial/

NumPy for MATLAB users

http://mathesaurus.sourceforge.net/matlab-numpy.html

Matplotlib Gallery

https://matplotlib.org/gallery.html

Gallery of Jupyter Notebooks

https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks

Introduction to Python 3 Notebooks

https://gitlab.erc.monash.edu.au/andrease/Python4Maths/tree/master

Acknowledgements

- The slides are created based on the educational materials for MATLAB from Kathleen Ossman and Gregory Bucks under BSD license.
- Supports from Texas A&M Engineering Experiment Station (TEES) and High Performance Research Computing (HPRC).

Appendix

ASCII Code

When you press a key on your computer keyboard, the key that you press is translated to a binary code.

```
A = 1000001      (Decimal = 65)
a = 1100001      (Decimal = 97)
0 = 0110000      (Decimal = 48)
```

ASCII Code

ASCII stands for **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Terminology

A **bit** is short for **bi**nary digi**t.** It has only two possible values: On (1) or Off (0).

A **byte** is simply a string of 8 bits.

A **kilobyte** (KB) is 1,024 (2^10) bytes.

A **megabyte** (MB) is 1,024 KB or 1,024^2 bytes.

A **gigabyte** (GB) is 1,024 MB or 1,024^3 bytes.

How Computers Store Variables

Computers store all data (numbers, letters, instructions, …) as strings of 1s and 0s (bits).

A **bit** is short for **bi**nary digi**t.** It has only two possible values: On (1) or Off (0).

Data Types: double and single

- A double uses 64 bits to store a real number.
- A single uses 32 bits to store a real number. Python does not support single by default though Numpy supports it.
- Doubles and singles can be used to represent both integers and non-integers.