



# Introduction to Programming with Python

---

Shuyang Liu, Swapnil Sayan Saha  
PhD Students, CS and ECE  
[sliu44@cs.ucla.edu](mailto:sliu44@cs.ucla.edu), [swapnilsayan@g.ucla.edu](mailto:swapnilsayan@g.ucla.edu)  
July 13, 2020

*Note: modified from original LACC slides by Prof. Lucas Wanner, Prof. Mani Srivastava, Dr. Mark Gottscho, Dr. Bharathan Balaji, Wojciech Romaszkan and Aishwarya Sivaraman*



**UCLA**

# Welcome to LACC 2020 @ UCLA!

---

## Introductions

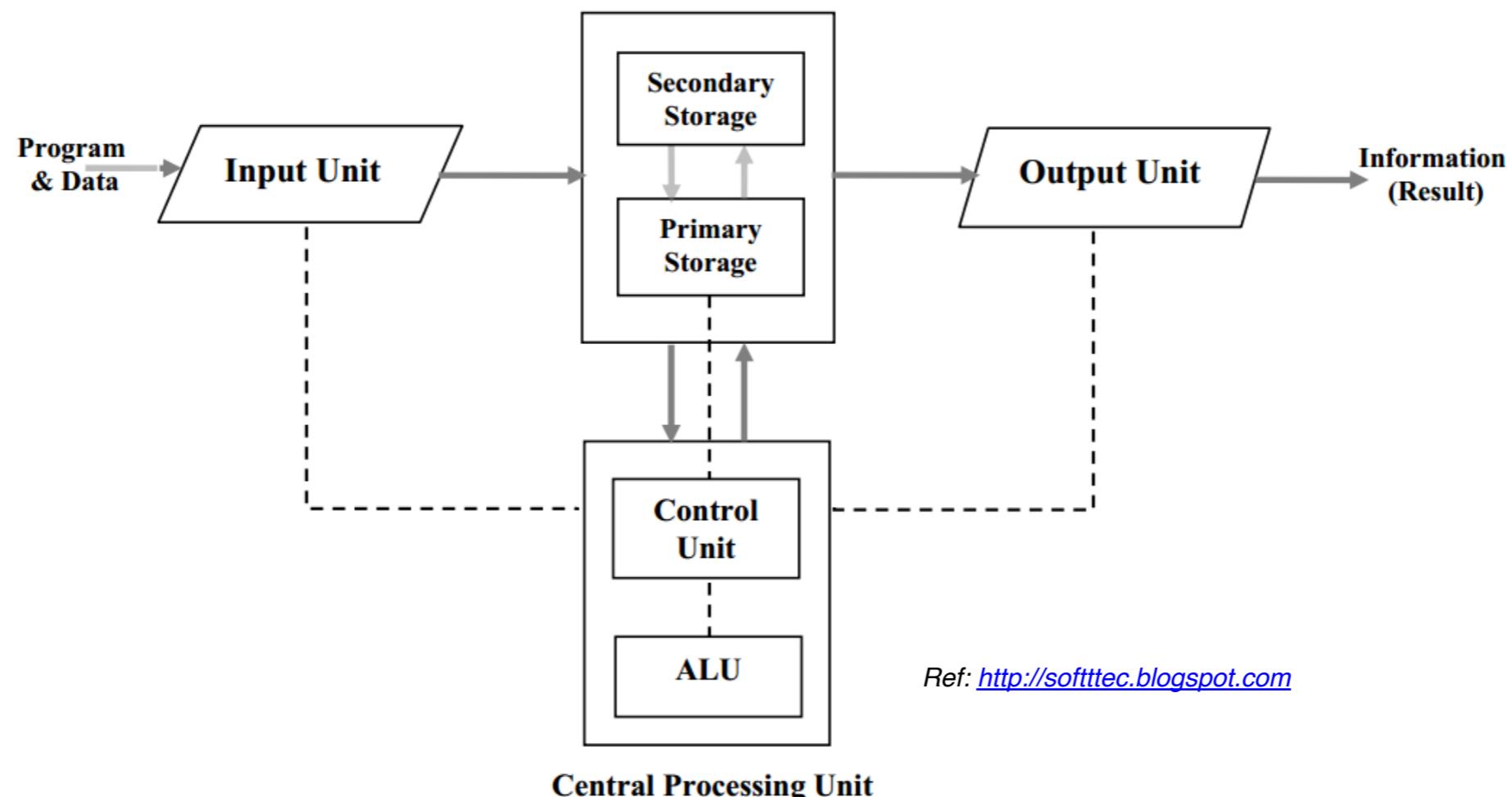
- Name?
- High school?
- Favorite activity outside school?
- Something interesting or different about you?
- What excites you about computing or engineering?



# Introduction to Computers

---

- A computer is an electronic **programmable** machine that can process almost all kinds of data. Fundamentally, a computer is a “**calculating machine**”.
- A computer performs **arithmetic** and **logical** operations.



# The Language of Computers

- A computer program is a **set of instructions** that dictates the steps required to perform a given task via some predefined **syntax** and **semantics**.
- A computer only understands 1s and 0s – **bits** (binary digits); 1 **byte** = 8 bits.

Decimal	Binary
1	0
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	'
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	\$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	:	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

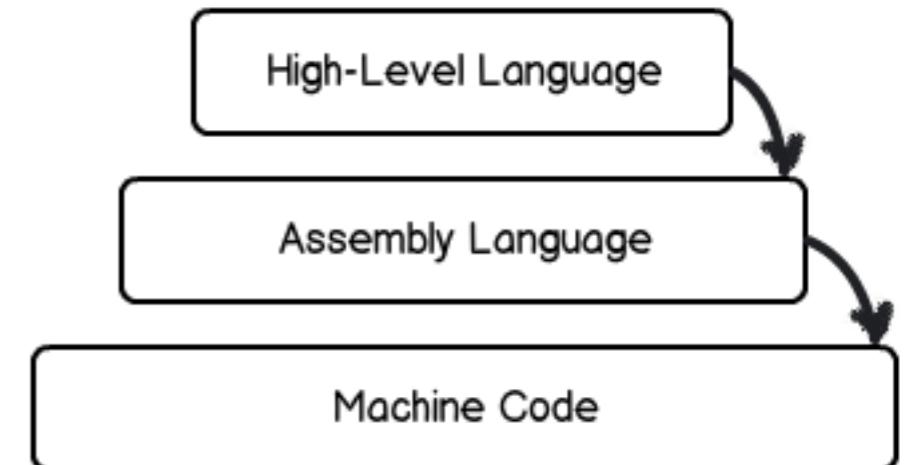
Ref: [http://web.alfredstate.edu/faculty/weimandn/miscellaneous/ascii/ascii\\_index.html](http://web.alfredstate.edu/faculty/weimandn/miscellaneous/ascii/ascii_index.html)



# Communicating with Computers

---

- Humans do not communicate in binary.
- We code in human-readable **programming languages**
  - Allow us to express programs in a way that is closer to natural languages
  - Provide libraries of commonly-used sub-tasks (functions)
  - **Translate high level code into machine code** through compilers or interpreters
- Given a language providing...
  - function definitions and calls, `f(x)`
  - a library function, `print("text")`
  - comparisons, `x == y`
  - assignments, `x = y`
  - conditional statements, `if (condition) ... else ...`



Ref: <http://shawnbiddle.com/js101/slides/class1.html#/6>

# Why do we choose Python?

---

- Relatively **simple syntax**.
- Relatively easy to **debug**.
- **Availability of open-source tools** and modules/libraries for eclectic applications with **large development community**.
- **Platform agnostic** (runs anywhere).
- **Broad application spectrum**, e.g. web applications / frameworks, standalone applications, video games, data science, artificial intelligence, machine learning, embedded systems etc.

# Getting and Learning Python

---

- **Python Official Website:** <http://www.python.org>
- Virtual environments with prebuilt Python modules:
  - **Anaconda:** <https://www.anaconda.com/products/individual>
  - **Jupyter:** <https://jupyter.org/install>
- Standalone Python IDEs:
  - **Pycharm:** <https://www.jetbrains.com/pycharm/>
  - **Thonny:** <https://thonny.org>
- Browser-based Python IDEs:
  - **Google Colab:** <https://colab.research.google.com>
  - **Trinket:** <https://trinket.io/python>
- Tutorials (basic syntax and semantics):
  - **Official Tutorial:** <https://docs.python.org/3/tutorial/>
  - **Tutorial by Bucky Roberts:**  
[https://www.youtube.com/playlist?list=PL6gx4Cwl9DGAcbMi1sH6oAMk4JHw91mC\\_](https://www.youtube.com/playlist?list=PL6gx4Cwl9DGAcbMi1sH6oAMk4JHw91mC_)
  - **Google:** <https://developers.google.com/edu/python/>



# Python Basics: Variables, Types & Operators

---

- **What is a Variable?**

A variable is a place in memory, which has a name, and where you can store a value (a number, phrase, etc.).

- **Assigning Values to Variables:**

- Python variables are declared automatically when you assign a value to a variable.
- The equal sign ( = ) is used to assign values to variables.

- **Values may be of different “types”**

```
In [1]: temperatureF = 78
```

- **Operators perform arithmetic and logical operations over variables and values.**



# Interacting with the External World (Users, and Files)

---

- Printing to the Screen (Hello World!)

```
>>> print "Python is really a great language,", "isn't it?";
Python is really a great language, isn't it?
```

- Reading Keyboard Input: *raw\_input([prompt])*

```
# file.py

str = raw_input("Enter your input: ");
print "Received input is : ", str
```

```
Enter your input: Hello Python
Received input is : Hello Python
```

- Reading and Writing Files

```
#!/usr/bin/python

# Open a file
fo = open("foo.txt", "wb")
fo.write( "Python is a great language.\nYeah its great!!\n");

# Close opend file
fo.close()
```

```
#!/usr/bin/python

# Open a file
fo = open("foo.txt", "r+")
str = fo.read(10);
print "Read String is : ", str
# Close opend file
fo.close()
```

Ref: <http://www.tutorialspoint.com/>



# Python Basics: Functions

---

- **Defining and Calling Functions:**

- A **function** is a block of organized, *reusable* code that is used to perform a single, related action.
- Functions provides better *abstraction* and *modularity* for your application and a high degree of code reusing
  - Makes your code simpler and easier to read!
- Calling a function: Once the basic structure is finalized, you can execute it by calling it from another function or directly from the Python prompt.



# Defining Functions

---

- By defining a function, we can put a routine into this function, and we will be able to execute this routine simply by calling this function.
- For example, we introduce a function “add5”, which add t onto the integer we passed into the function.
- Example: In [58] : 

```
def add5(x):
    return x+5
```
- In this function, x is the integer we passed into the function. And the function will return the value of x+5 back.



# Calling the Function

---

- To call the function add5 we just defined, we simply do:

```
In [59]: result = add5(10)  
result
```

```
Out[59]: 15
```

- By passing number 10 into the add5 function, 5 is added.
- The result then equals to 15.

❖ **PYTHON PROGRAMMING CAN BE YOUR CALCULATOR!**



# Exercise A – Printing to the Screen

---

- Open the Jupyter Notebook in Google Colab.
- We will discuss the section titled “The Very First Step” (Exercise 0).
- We will complete the section titled “The Very First Step” (Exercise 0).



# Python Operators

---

## □ What is an operator?

- Python operators are very similar to the mathematical operators we use everyday. For example, +, -, \*, ...



# Python Basic Operators

---

- **Python Operator Types:**

- Arithmetic Operator
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operator



# Python Arithmetic Operators

---

Assume variable **a** holds 10 and variable **b** holds 20 then:

Addition	$a + b = 30$
Subtraction	$b - a = 10$
Multiplication	$a * b = 200$
Division	$b / a = 2$
Reminder	$b \% a = 0$
Exponent	$a ** b = 10^{20}$

```
In [3]: temperatureC = (temperatureF - 32)*0.5556
print(temperatureC)
25.5576
```



# Python Comparison Operators

---

Assume variable **a** holds 10 and variable **b** holds 20 then:

Equality	$a == b = \text{False}$
Inequality	$a != b = \text{True}$
Less than	$a < b = \text{True}$
Greater than	$a > b = \text{False}$
Less than or equal	$a <= b = \text{True}$
Greater than or equal	$a >= b = \text{False}$

```
In [4]: temperatureF > 80
Out[4]: False
```



# Python Logical Operators

---

Assume variable **a** holds True and variable **b** holds False then:

Logical AND	<code>a and b = False</code>
Logical OR	<code>a or b = True</code>
Logical NOT	<code>not a = False</code>

```
In [9]: humidity = 50  
        temperatureF > 80 or humidity > 30
```

```
Out[9]: True
```



# Python Data Types

---

- In computer programming, a data type is a classification identifying one of various types of data, just like in real life we separate words “Hello!” and numbers “12345”.
- **Standard Data Types in Python:**
  - **Numbers**, e.g., 1000
  - **String**, e.g., “Hello world!”
  - **Boolean** e.g. True
  - **List**, a container e.g., [365, 12, 30]
  - **Dictionary**, a key-indexed container.



# Python Types: Numbers

---

- **Numbers**

- Store numeric values,

- Example:

```
In [1]: width = 20  
height = 15  
width * height
```

Out[1]: 300

- In this example, “width = 20” defines variable width to be 20, and “height = 15” defines height to be 15. This sort of number is called **integer** or **int**.

- After initializing width and height, we can calculate width \* height as shown in the example.

- To use real numbers (also called **float** / **double**), add a dot:

```
In [10]: width = 20.5  
height = 15.25  
width * height
```

Out[10]: 312.625



# Python Types: Lists

---

- **Lists:**
  - Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets.

```
In [2]: a = ['blue', 'red', 100, 365]
a
```

```
Out[2]: ['blue', 'red', 100, 365]
```
  - Example:
    - In this example, we assign four items to list a. The items are string ‘blue’, string ‘red’, number 100, and number 365.
    - We print a’s items in the second command, and get its four items printed.
    - Note that items in the list have different types (string and int) – this is not common among programming languages.



# Python Types: Lists

---

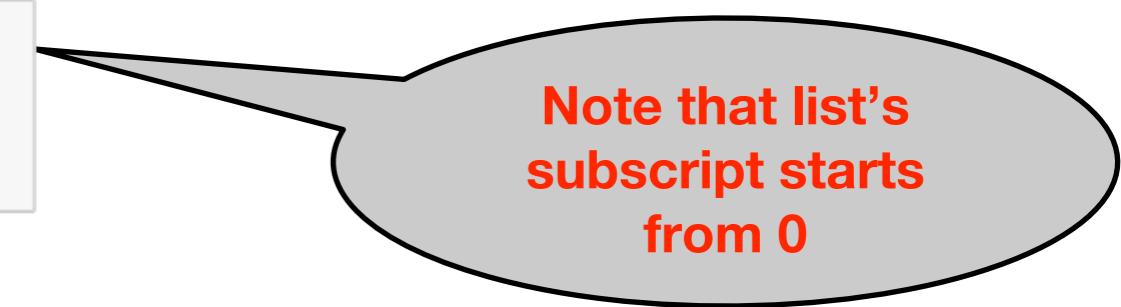
- **Lists:**

- It is possible to change individual elements of a list.

- Example:

```
In [7]: a = ['blue', 'red', 100, 365]
          a[2] = a[2] + 50
          a[2]

Out[7]: 150
```



Note that list's  
subscript starts  
from 0

- In this example we modify the **third** element in the list by adding 50 to its value
  - We then print the third element by using *name[index]*



# Python Types: Lists

---

- **Lists:**
  - Many “operations” and “methods” available for lists:

Is the value in the list?	<code>100 in a = True</code>
Is the value not in the list?	<code>'yellow' not in a = True</code>
Length of the list	<code>len(a) = 4</code>
Index of an element	<code>a.index('red') = 1</code>
Append element	<code>a.append(10)</code>
Reverse the list	<code>a.reverse()</code>

+ sort, concatenate, insert, sum and many more at:

<https://docs.python.org/2/tutorial/datastructures.html#more-on-lists>



# Python Types: Range

---

- **Creating number sequences:**

- **range(n)** for integer n > 0 returns the list [0, 1, ... n-1]

```
In [26]: a = range(10)
          print(*a)
```

```
0 1 2 3 4 5 6 7 8 9
```

- **range(m,n)** returns the list [m, m+1, ... n-1]

```
In [31]: a = range(5,10)
          print(*a)
```

```
5 6 7 8 9
```

- **range(m,n,o)** returns the list [m, m+o, ... n-1]

```
In [32]: a = range(0,10,2)
          print(*a)
```

```
0 2 4 6 8
```



# Exercise B – Lists, Variables and Data Types

---

- Open the Jupyter Notebook in Google Colab.
- We will discuss the section titled “Variables and Types” (Exercise 1, 2 and 3).
- We will complete the section titled “Variables and Types” (Exercise 1, 2 and 3).



# Python Types: Strings

---

- **Strings:**

- Strings in Python are identified as a contiguous set of characters in between quotation marks.

```
In [34]: string = "ABCDEFG."
string
```

```
Out[34]: 'ABCDEFG.'
```

- In this example, we assign “ABCDEFG.” to variable string. And we print string.
- The result we get is still “ABCDEFG.”
- You can use double “ or single ‘ quotation marks for strings in Python, they are equivalent.



# Python Types: Strings

---

- **Strings:**
  - Strings can be subscripted (indexed). This is because they are essentially lists of characters.
  - Example: In the string defined as str = “ABCDEFG.”, we query the **fifth** letter in this string.

```
In [35]: string = "ABCDEFG."
          string[4]
```

```
Out [35]: 'E'
```

Because it's a list,  
first letter has index  
0



# Python Types: Strings

---

- **Strings:**
  - Many “operations” and “methods” available for strings:

Is the letter in the string?	‘A’ in string = True
Is the substring in the string?	‘CDE’ in string = True
Length of the string	<code>len(string) = 8</code>
Index of a letter (first)	<code>string.index('A') = 0</code>
Append element	<code>string + "Z" = "ABCDEFG.Z"</code>
Make lowercase	<code>string.lower() = "abcdefg."</code>

and many more at

<https://docs.python.org/2/library/stdtypes.html#string-methods>



# Python Types: Dictionaries

- **Dictionaries:**
  - Represents a set of key-value pairs where keys are unique (i.e. numbers, strings) but values need not be. It's not ordered.
  - Example:

```
In [54]: D = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print ('Name: ' + D['Name'])
print ('Age: ' + str(D['Age']))
```

Name: Zara
Age: 7
  - Updating a dictionary

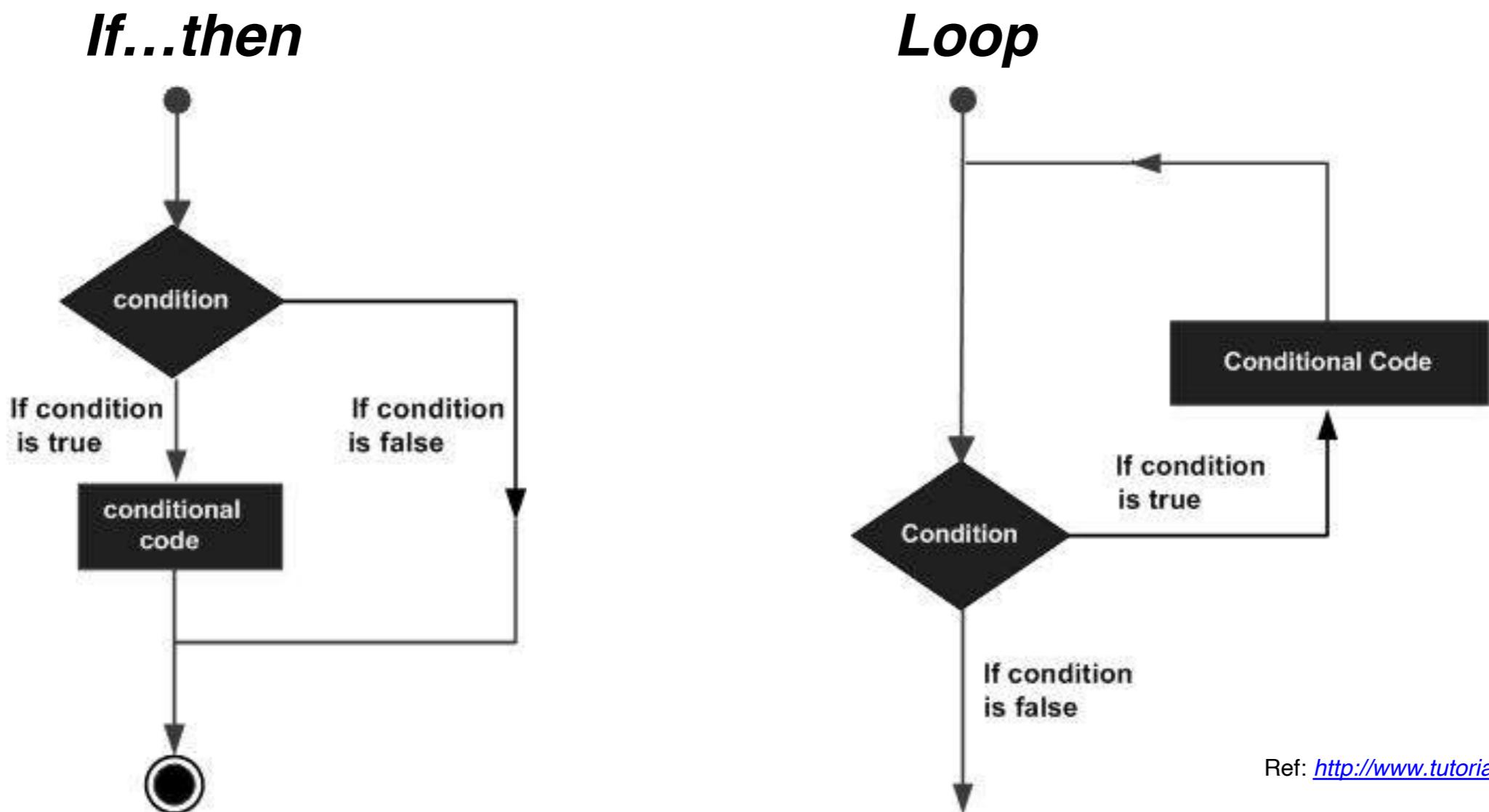
```
In [56]: D = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
D['Age'] = 8 # Update entry
D['School'] = 'UCLA' # Add new entry
D
```

```
Out[56]: {'Age': 8, 'Class': 'First', 'Name': 'Zara', 'School': 'UCLA'}
```



# Beyond Sequential Execution: Control Flow Tools

- Programming languages provide various control structures that allow for more complicated execution paths instead of just sequential
  - *Make decisions on what to do dynamically while the program runs!*



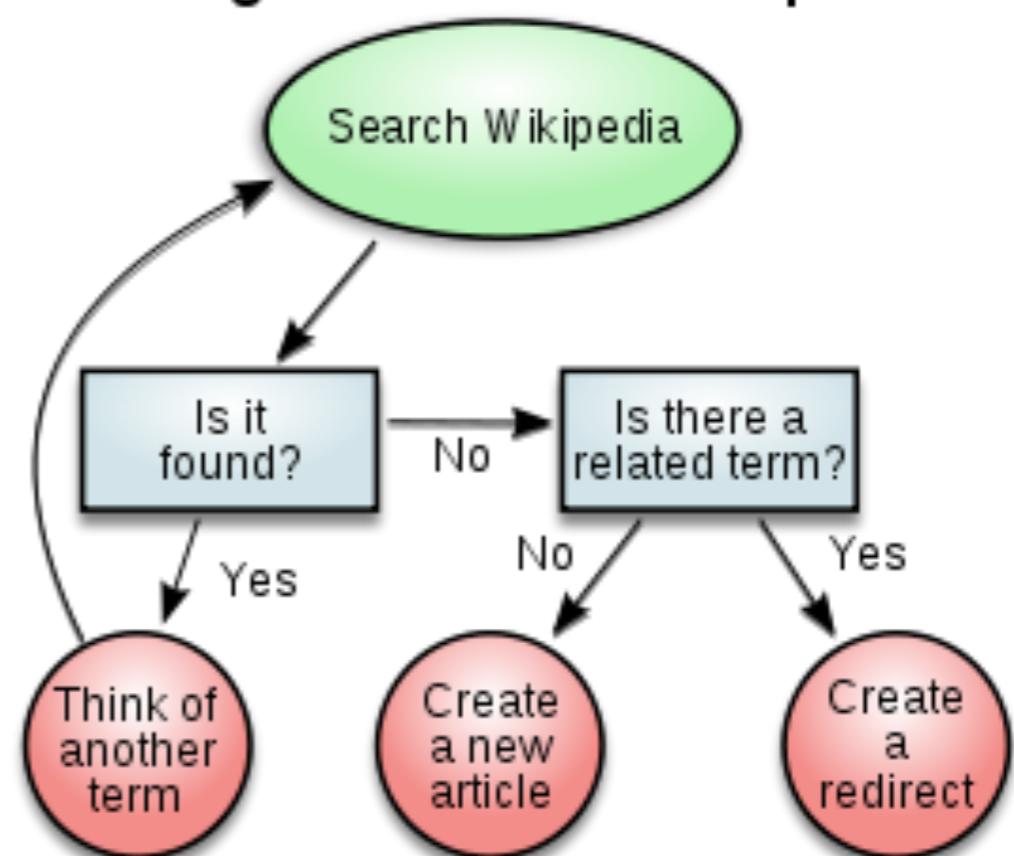
# Python Basics: Decision Making

---

- **Decision Making:**

- Conditional constructs are used to incorporate decision making into programs.
- The result of this decision making determines the sequence in which a program will execute **instructions**.

**Adding an article to Wikipedia**



# Python Basics: Decision Making

---

- **if Statement**
  - The **if** statement contains a logical expression using which data is compared, and a decision is made based on the result of the comparison.

- Example:

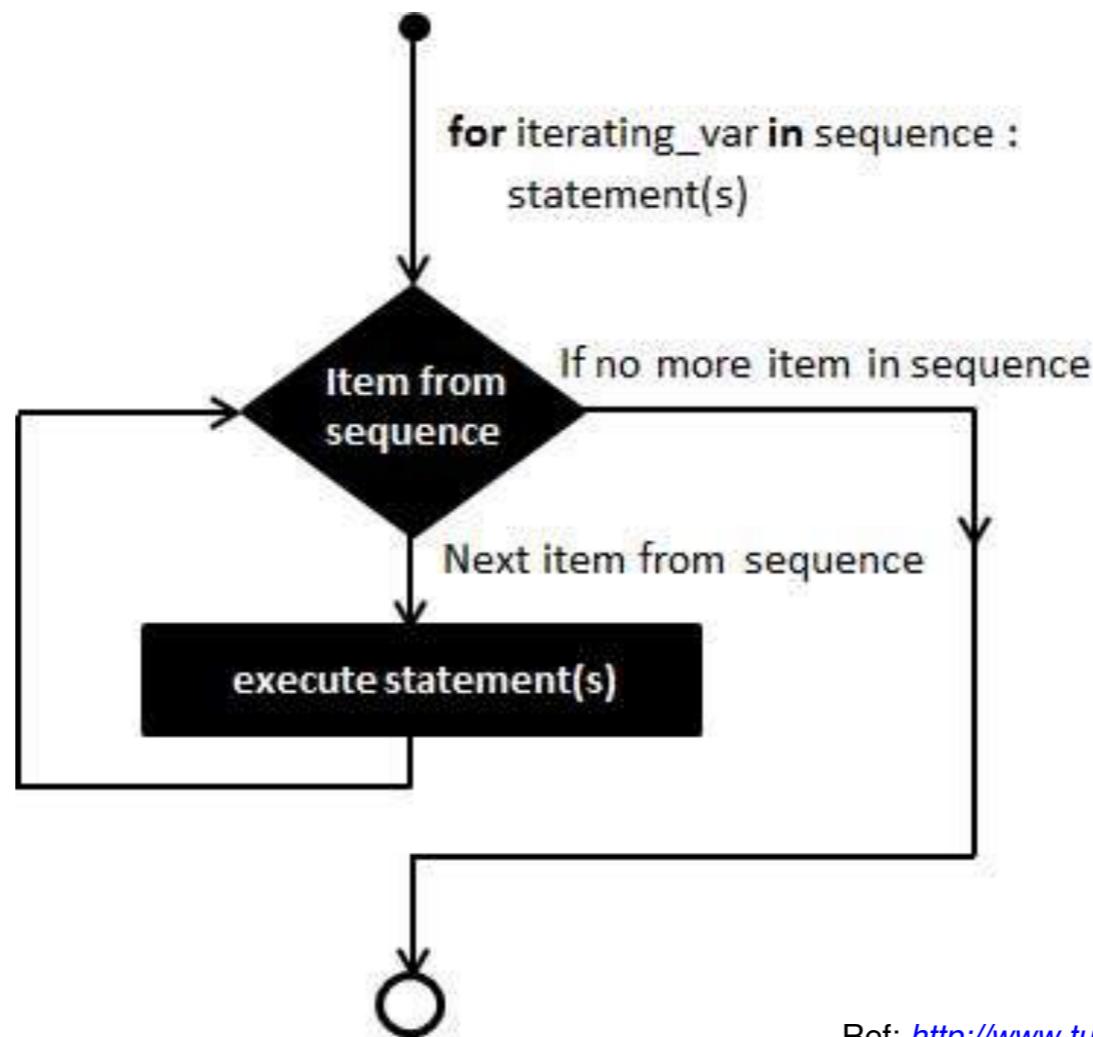
```
>>> x = int(raw_input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...     x = 0
...     print 'Negative changed to zero'
... elif x == 0:
...     print 'Zero'
... elif x == 1:
...     print 'Single'
... else:
...     print 'More'
...
More
```

- In this example, we input integer 42, which is not less than 0, not equal to 0, and not equal to 1. Therefore, it falls into the option that “print ‘More’”.
- Any non-zero and non-null values are treated as TRUE by Python



# Python Basics: Looping

- **for statement**
  - The **for** loop in Python has the ability to iterate over the items of any sequence, such as a list, tuple, or a string.



Ref: <http://www.tutorialspoint.com/>



# Python Basics: Looping

---

- **for statement**
  - The **for** loop in Python has the ability to iterate over the items of any sequence, such as a list or a string.
  - Example:

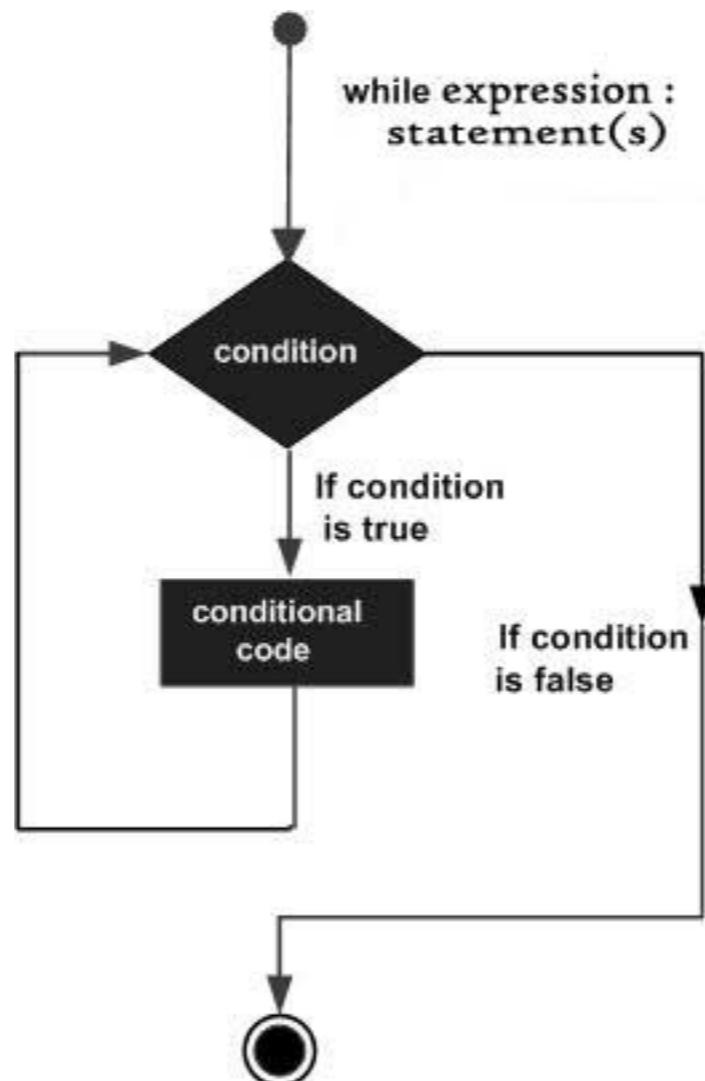
```
>>> # Measure some strings:
... a = ['cat', 'window', 'defenestrate']
>>> for x in a:
...     print x, len(x)
...
cat 3
window 6
defenestrate 12
```
  - In this example, all the items in list a get printed.
  - `len(x)` is a built-in function in Python, which returns the length of x.
  - In this example, x is a string, such as “cat”. `len(x)` returns “cat”’s length, which is 3.



# Python Basics: Looping

---

- **while statement**
  - The **while** loop in Python has the ability to iterate over a statement or group of statements while a given condition is TRUE



Ref: <http://www.tutorialspoint.com/>



# Python Basics: Looping

---

- **while statement**
  - The **while** loop in Python has the ability to iterate over a statement or group of statements while a given condition is TRUE

- Example:

```
count = 0
while (count < 9):
    print 'The count is:', count
    count = count + 1

print "Good bye!"
```

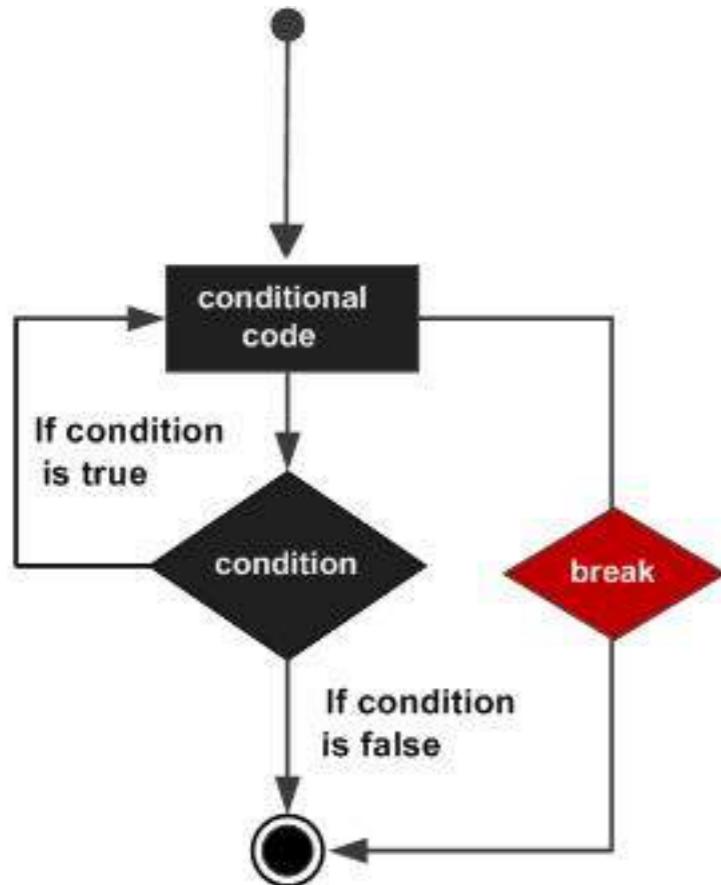
```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

Ref: <http://www.tutorialspoint.com/>

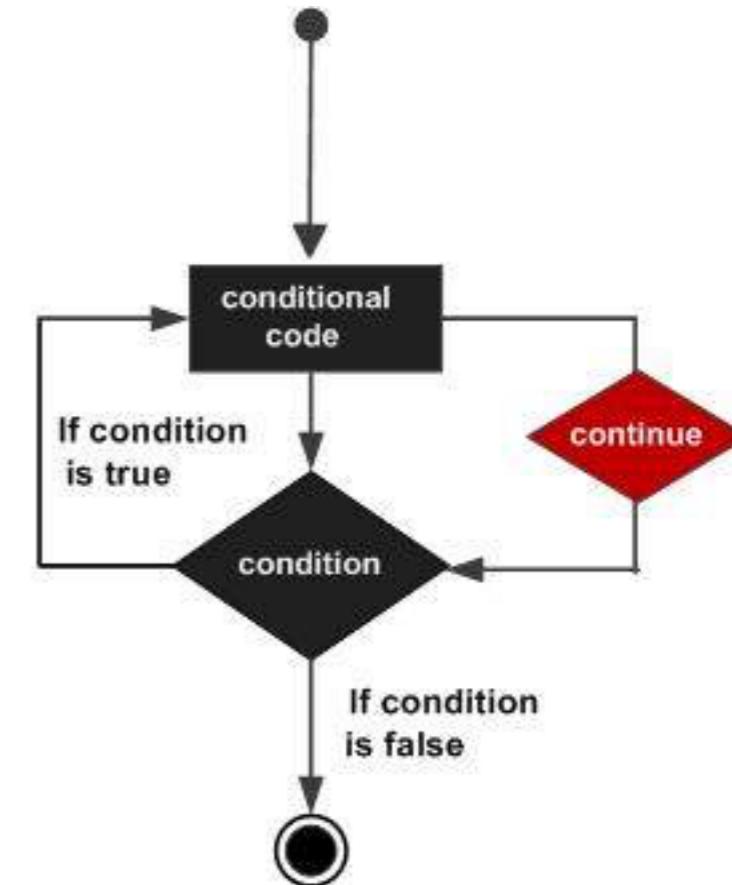
- Caution: “infinite loop” if a condition never becomes false.



# Python Basics: Loop Control Statements



```
for letter in 'Python':  
    if letter == 'h':  
        break  
    print 'Current Letter :', letter
```



```
for letter in 'Python':  
    if letter == 'h':  
        continue  
    print 'Current Letter :', letter
```

Ref: <http://www.tutorialspoint.com/>



# Exercise C – Looping

---

- Open the Jupyter Notebook in Google Colab.
- We will discuss the sections titled “A Simpler way to print: Using for loops”, “Interacting with the outside world: I/O” and “if Checks” (Exercise 4, 5 and 6).
- We will complete these sections as well (Exercise 4, 5 and 6).



# Python Modules and Namespaces

---

- A **module** is a file consisting of Python code which can define functions, variables etc.
  - The code for a module named *mname* normally resides in the file *mname.py* which is searched for in selected folders on the computer
  - Any Python source file can be used as a module by executing an **import** statement in some other Python source file.

```
# file: support.py
def print_func( par ):
    print "Hello : ", par
    return

# some other file
import support
# Now one can call function defined in that module as follows
support.print_func("Zara")
```

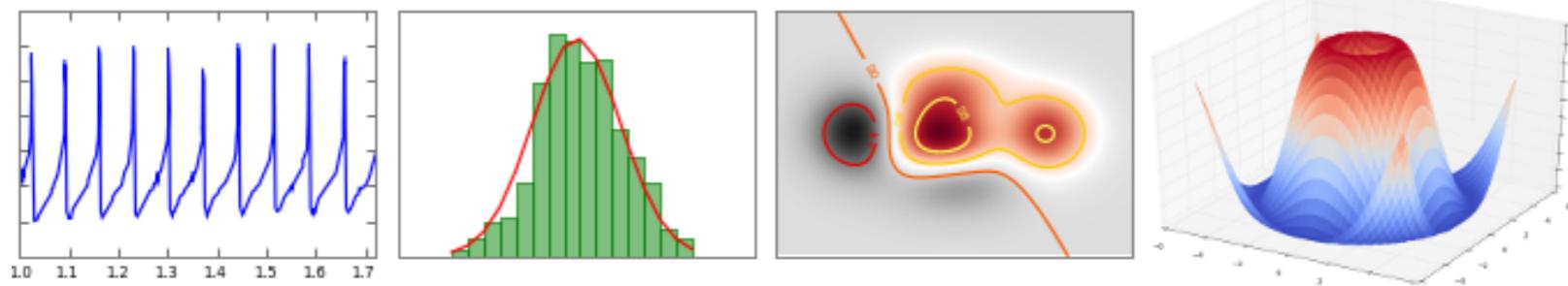
- To prevent confusion with same variable and function names being used in different modules, they belong to a module's **namespace** which is a dictionary of variable names (keys) and their corresponding objects (values).
- Each function has its own *local namespace*, and a Python statement can directly access variables in the *local namespace* and in a *global namespace*.



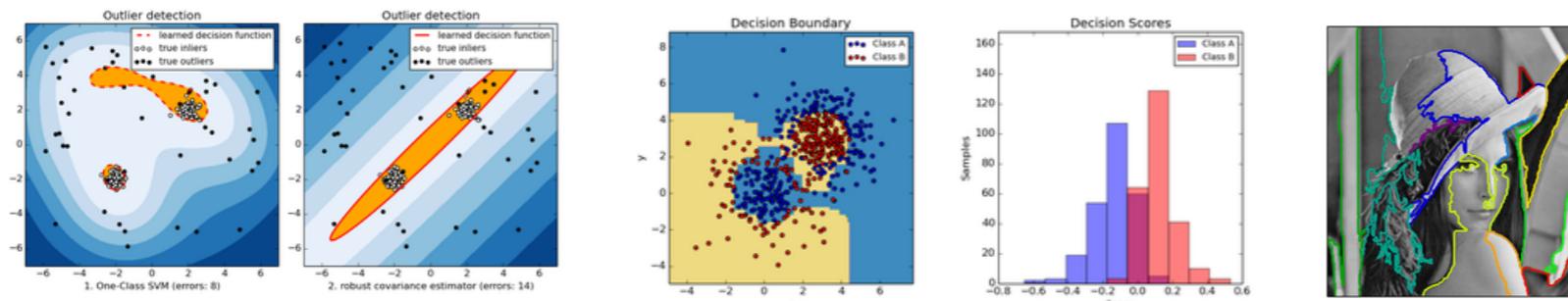
# Some Very Useful Python Modules

---

- **matplotlib:** a python plotting library



- **NumPy:** computing with n-dimensional arrays
- **Tensorflow:** AI, machine learning and data science
- **SciPy:** data science, numerical integration and optimization
- **Sympy:** symbolic mathematics
- **scikit-learn:** data mining and machine learning



- **pygame, pyget:** games
- and many many more...

# Exercise D – Wrapping Up

---

- Open the Jupyter Notebook in Google Colab.
- We will discuss the section titled “The Game Loop: while loop” (Exercise 7).
- We will complete the section as well. (Exercise 7).

