



Introduction to Programming with Python

Wojciech Romaszkan
ECE PhD Student
wromaszkan@ucla.edu
July 8-9, 2019

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



UCLA

Welcome to LACC 2019 @ UCLA!

- Introductions

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



Introduction

- What are the similarities and differences between a calculator and a computer?



© 2008 CNET Networks, Inc.

The ***fundamental*** difference between a computer and a calculator is that computers are ***programmable!***



What does “**programmable**” mean?

A program is a “script” of operations stored in memory that tells a computer what to do.

Think of a program as a recipe for solving a problem. It’s a step-by-step procedure.

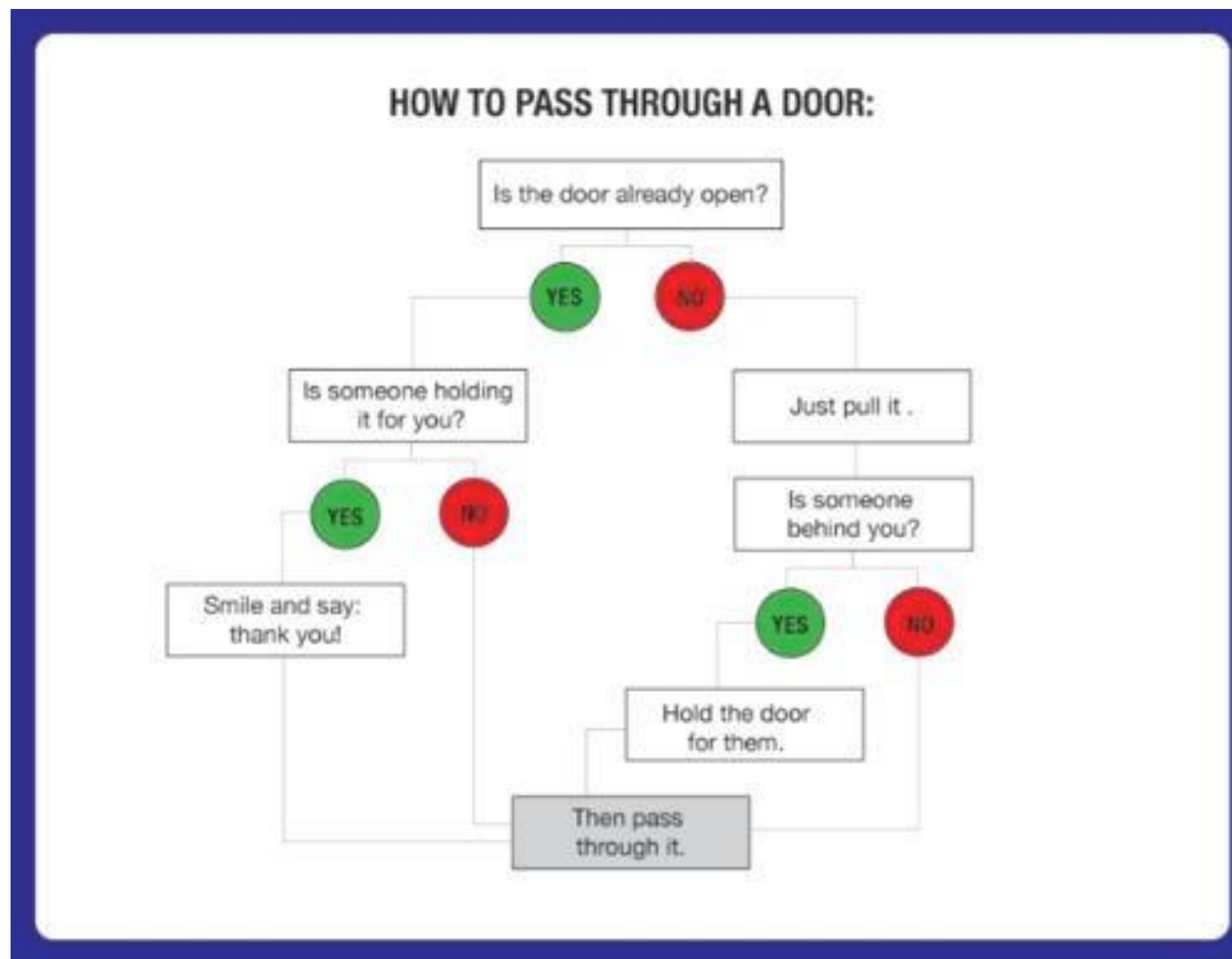
- A calculator can...
 - perform arithmetic operations
 - ~~perform logic operations~~
 - store values in memory
 - load values from memory
- A computer can...
 - perform arithmetic operations
 - perform logic operations
 - store values in memory
 - load values from memory

Unlike a calculator, a computer can make choices about what to do.
These are made using branches and loops in a procedure.
Essentially, decisions on what computation to do based on a condition.



Conditional Branches

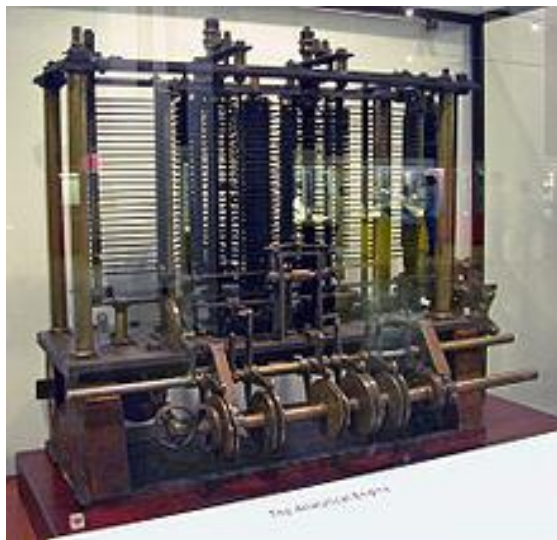
“A decision on which path to take based on some condition”



A Bit of History:

Who invented the programmable computer?

- Charles Babbage originated the concept of and designed a mechanical programmable computer in 1837 called the “Analytical Engine”, but couldn’t finish it as the British Government ceased funding.
- The first programmable computers were not actually built until ~ 1940 using electromechanical relays (by German engineer Konrad Zuse)
- World's first electronic digital programmable computer (using vacuum tubes) was built by Eckert and Mauchly at U. Penn, but invention attributed to Atanasoff of Iowa State



Reconstruction of Babbage's Analytical Engine, the first general-purpose programmable computer.



Replica of Zuse's Z3, the first fully automatic, digital (electromechanical) general-purpose programmable computer.



ENIAC, the first electronic general-purpose computer

How can we express a program?

Using real machine code (binary)?

1: “on”

0: “off”

... 10101001000000101011111 0010101 00011 10100101001 0100 100100100 0000000000 ...

How do we humans do anything with this? Surely a better way...

How about machine language
(**assembly code**) instead?

Code comments — human-readable
explanations that don't do anything in the program



```
pass_door:
    ld register_1, door_status
    ld register_2, door_open
    beq register_1, register_2, holding
    call pull_door
```

```
holding:
    ...
```

```
pull_door:
    ...
```

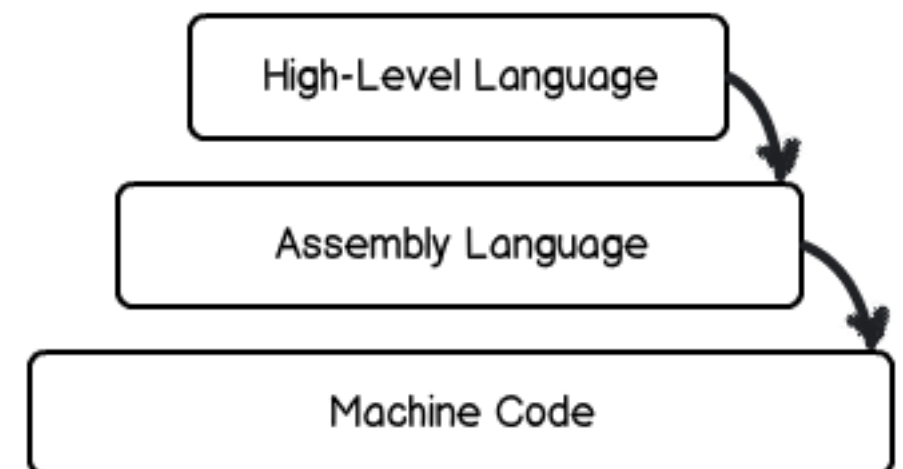
```
# load contents from memory into register

# if register 1 and 2 are
# equal, go to (branch to) label
# "holding"
```



Can we do better still?

- We actually code in high-level, 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 ...`
- ...how can we write the “pass door” program?



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

Part of the “Pass door” program in “Pseudocode”

```
function pass_door()
    if (door_status == door_open)
        if (hold_status == someone_holding)
            print("Thank you")
        else
            pull_door()

function pull_door()
    door_status = door_open
    ...
```

function definition
conditional statements,
comparisons
call library function

call function

assignment of value to variable



Programming in Python

- **Why can't we program in English?**
- Why do we need so many programming languages?
- Why Python?



Why do we choose Python?

- **Python has simple syntax.**
 - English has straightforward grammar.
- **Python comes with a great set of tools.**
 - English has abbreviations and idioms.
- **Python programs can be run on Linux, MAC, etc.**
 - English is spoken by people from the whole wide world.
- **Python is relatively easy to debug.**
 - English mistakes are easy to be found.



How simple is Python?

- To print “Hello world!” on screen, C, Java and Python need to do:

- **C:**

```
#include <stdio.h>

int main() {
    printf("Hello World!");
    return 0;
}
```

- **Java:**

```
public class HelloWorld {

    public static void main (String[] args) {
        System.out.println("Hello, world!\n");
    }

}
```

- **Python:** `print "Hello, World!"`



Find Python around you

- Python is used everywhere!
 - **Web applications (server side):**
 - YouTube
 - Facebook
 - Dropbox (original)
 - Pinterest
 - MoinMoin, a popular wiki engine



Find Python around you

- Python is used everywhere!
 - **Applications (client side):**
 - Original BitTorrent client, along with several derivatives
 - Ubuntu Software Center, a graphical package manager
 - Dropbox (desktop application)
 - Mercurial



Find Python around you

- Python is used everywhere!

- ☐ **Web frameworks:**

- ☐ Google App Engine
- ☐ Django



Find Python around you

- Python is used everywhere!

- ☐ **Video games:**

- ☐ Civilization IV
- ☐ Battlefield 2



Getting Python

- **Python Official Website:** <http://www.python.org>
- Lots of “packages” that add capabilities to the basic language and “tools” that make writing, running, and debugging easier
- Free and commercial “distributions” that bundle core Python with useful packages and tools and simplify installation, e.g.
 - **Anaconda:** <https://store.continuum.io/cshop/anaconda/>
 - Enthought Canopy: <https://www.enthought.com/products/canopy/>
- Browser-based Python (limited functionality)
 - http://www.tutorialspoint.com/ipython_terminal_online.php
 - http://www.tutorialspoint.com/execute_python_online.php
 - <http://repl.it/languages/Python>
 - <http://pythonfiddle.com>



Free Python Learning Material on the Web

- Official Tutorial: <https://docs.python.org/2/tutorial/>
- TutorialsPoint: <http://www.tutorialspoint.com/python/index.htm>
- Google: <https://developers.google.com/edu/python/>
- LearnPython: <http://www.learnpython.org>
- Code Academy: <http://www.codecademy.com/en/tracks/python>
- And many many more...



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
```

- **Operations over Variables and Values**



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	<code>a == b</code> = False
Inequality	<code>a != b</code> = True
Less than	<code>a < b</code> = True
Greater than	<code>a > b</code> = False
Less than or equal	<code>a <= b</code> = True
Greater than or equal	<code>a >= b</code> = False

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



Python Logical Operators

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

Logical AND	a and b = False
Logical OR	a or b = True
Logical NOT	not a = False

```
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.
- After initializing width and height, we can calculate width * height as shown in the example.
- To use real numbers, add a dot:

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

```
Out[10]: 312.625
```



Warm-up

- ☐ Start up the Jupyter notebook
- ☐ Do the “Warm-up” part (instructions included)
- ☐ Feel free to ask questions



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.

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

`Out[2]: ['blue', 'red', 100, 365]`

- 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 $[0, 1, \dots, n-1]$

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

0 1 2 3 4 5 6 7 8 9

- **range(m,n)** returns the list $[m, m+1, \dots, n-1]$

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

5 6 7 8 9

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

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

0 2 4 6 8



Exercise # 1

- Go to Exercise 1 section in your Jupyter Notebook
- Given a list `L = [2, 3, 4, 5, 6]`
 - Print the second item in this list.
 - Get the sum of all the values in this list.
 - Append value 7 after value 6 into the list.
- Look at <https://docs.python.org/2/tutorial/datastructures.html#more-on-lists> for various methods available for list data type



Python Types: Strings

□ Strings:

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

- Example:

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

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

- In this example, we assign “ABCDEFGG.” to variable string. And we print string.
- The result we get is still “ABCDEFGG.”
- 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 = "ABCDEFGH."`, we query the **fifth** letter in this string.

```
In [35]: string = "ABCDEFGH."  
         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	len(string) = 8
Index of a letter (first)	string.index('A') = 0
Append element	string + "Z" = "ABCDEFGG.Z"
Make lowercase	string.lower() = "abcdefghg."

and many more at

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



Exercise # 2

- Go to Exercise 2 section in your Jupyter Notebook
- Given a string S: “This is an example string”
 - Print the fifth letter in the string.
 - Capitalize all letters.
 - Replace spaces with hyphens (a bit more advance).
 - Look at <https://docs.python.org/2/library/stdtypes.html#string-methods> for various methods available for list data type



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'}



Exercise # 3

- ❑ Go to Exercise 3 section in your Jupyter Notebook
- ❑ Given two dictionaries, D and E, do the following:
 - Add John's and Tom's surnames with "Surname" as a key.
 - Change John's city to NY.
 - Create a list F with both dictionaries as it's elements.
 - Print out the age of the first person on the list F.
- Look at <https://www.programiz.com/python-programming/methods/dictionary> for various methods available for dictionary data type



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!**



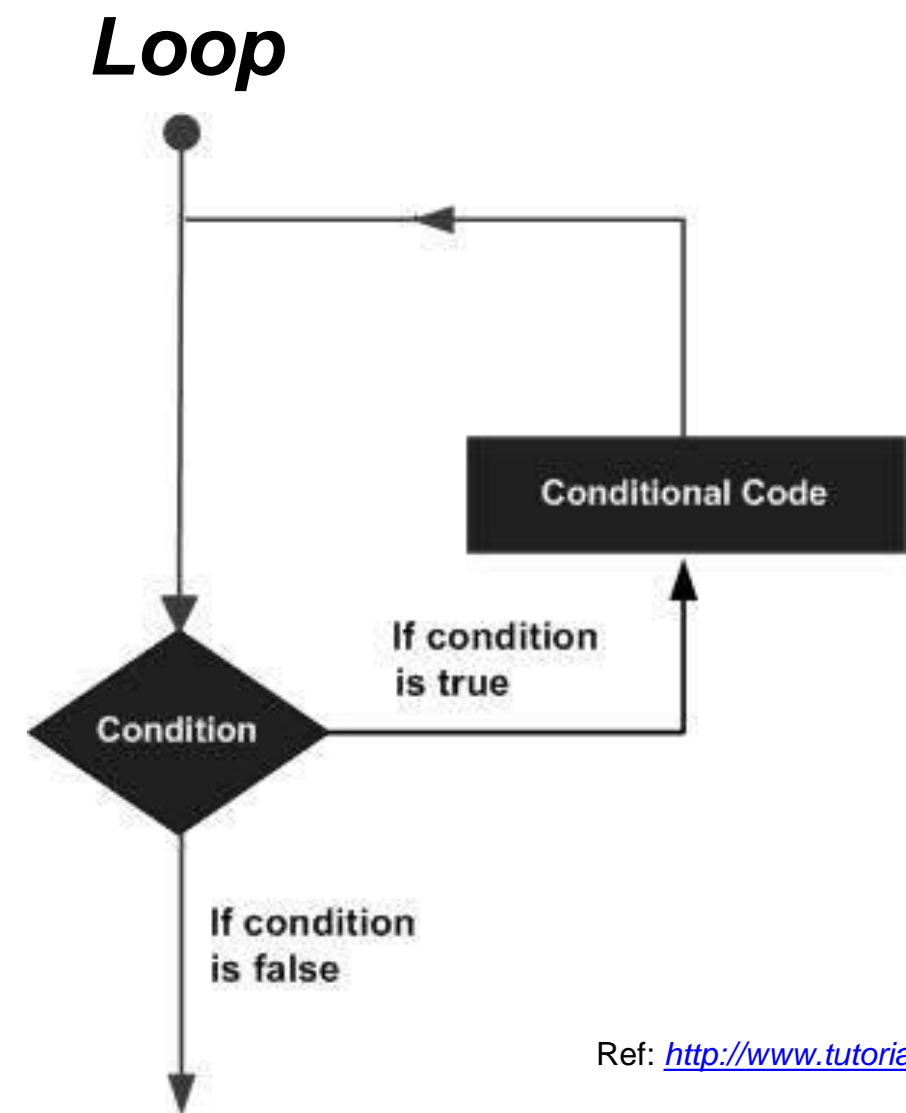
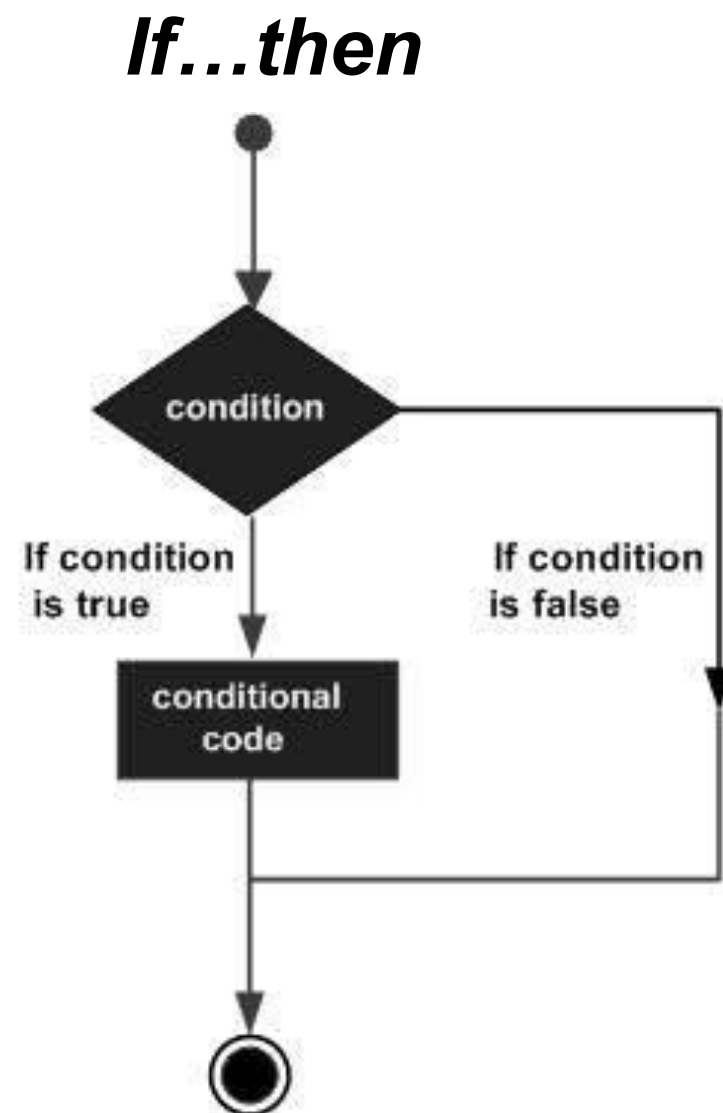
Exercise # 4

- Write a function that does the following things:
 - The function takes a number x into the function.
 - In the function, deduct 10 from x.
 - If the result of the deduction is greater than equal to 0, the function returns 1.
 - If the result of the deduction is less than 0, the function return 0.



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!

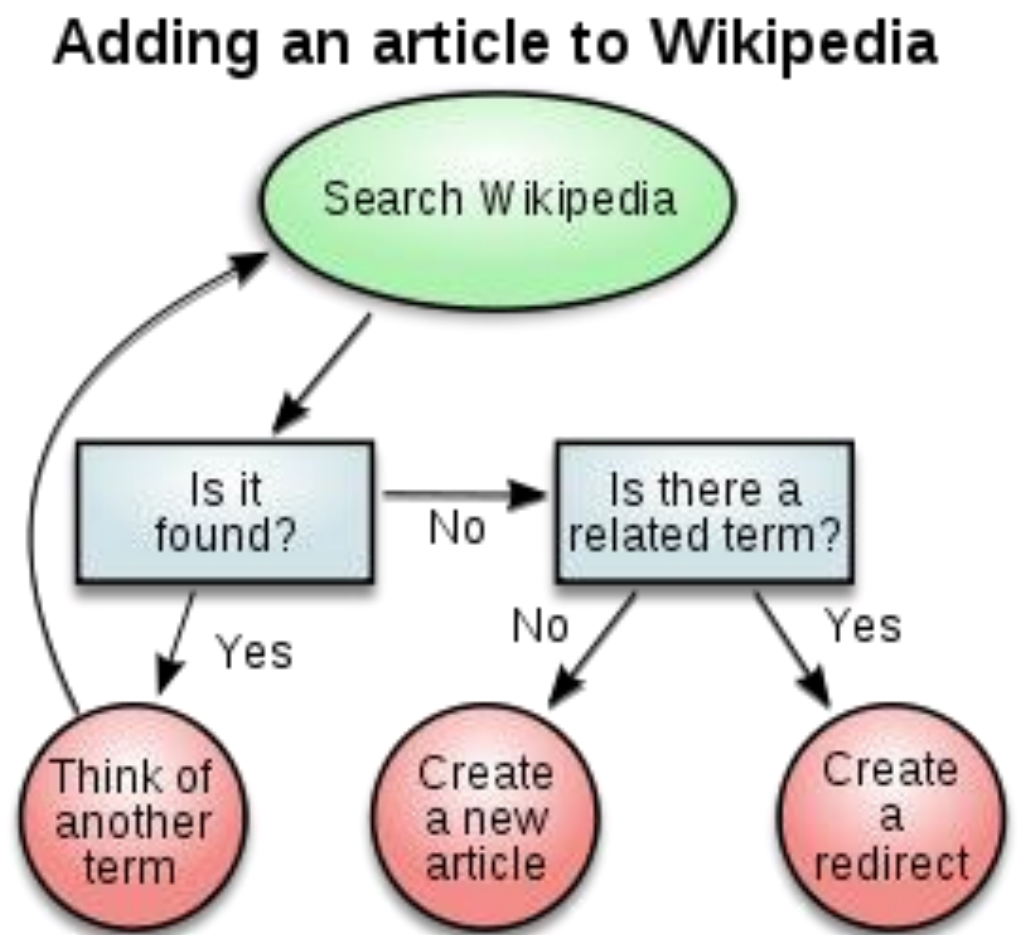


Ref: <http://www.tutorialspoint.com/>

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**.



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
```

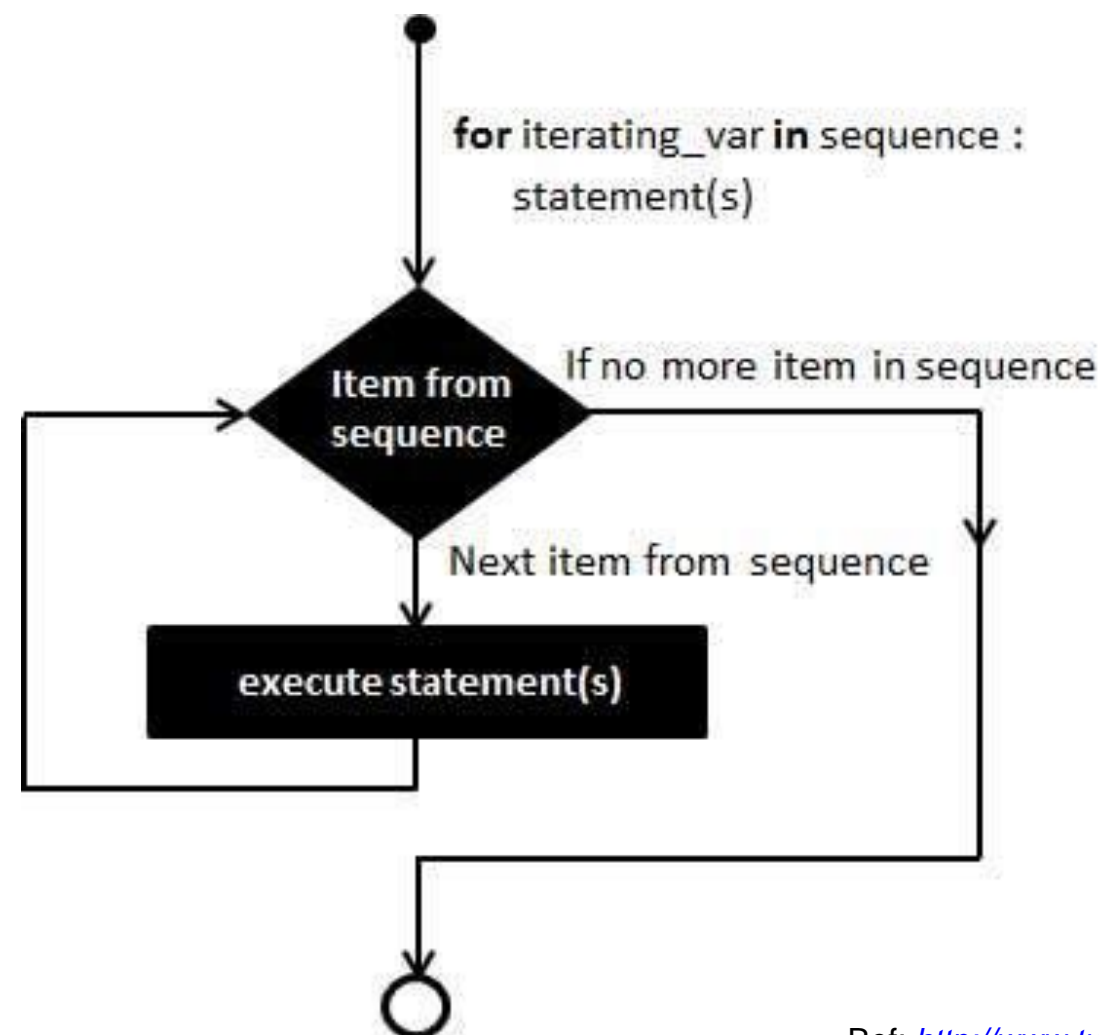
- In this example, we input integer 42, which is not less than 0, not equal to 0, and not equal to 1. Therefore, it falls into the option that “print ‘More’”.
- Any non-zero and non-null values are treated as TRUE by Python



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
```

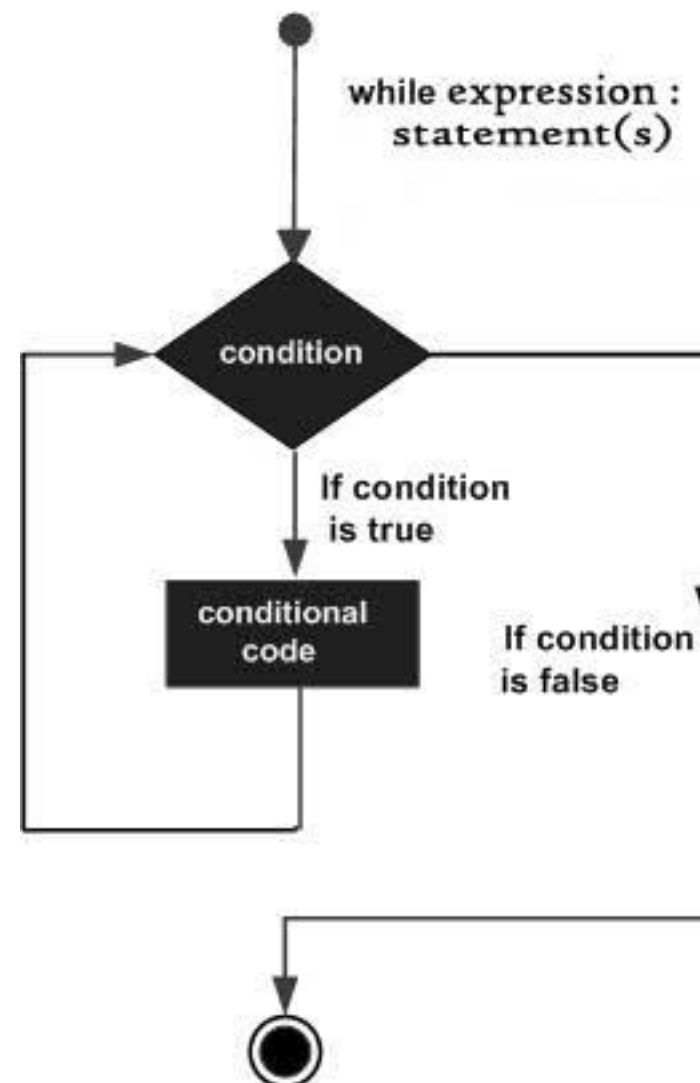
- In this example, all the items in list `a` get printed.
- `len(x)` is a built-in function in Python, which returns the length of `x`.
- In this example, `x` is a string, such as “cat”. `len(x)` returns “cat”’s length, which is 3.



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!"
```

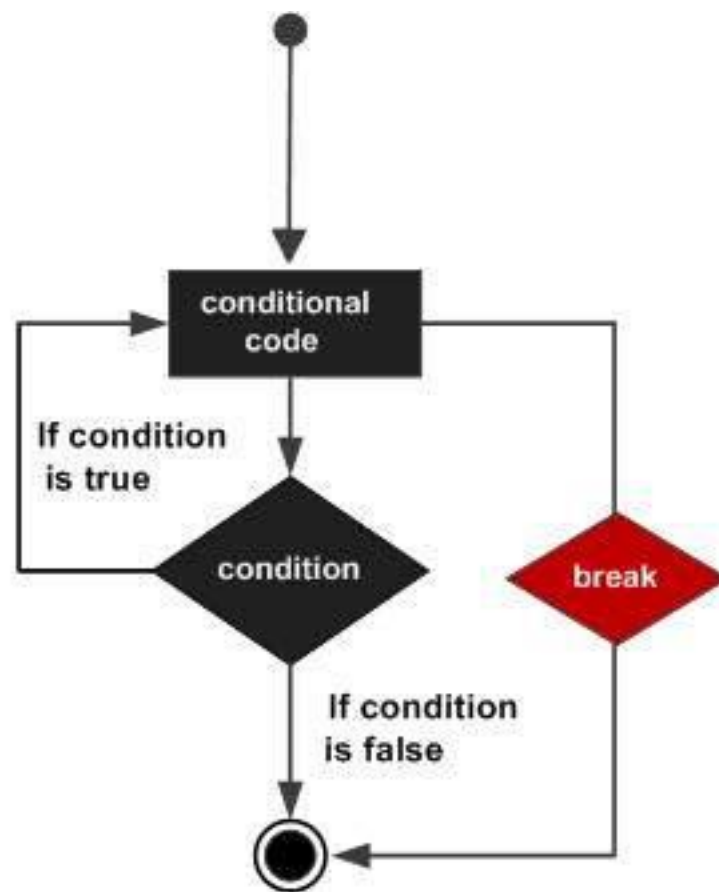
```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

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

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

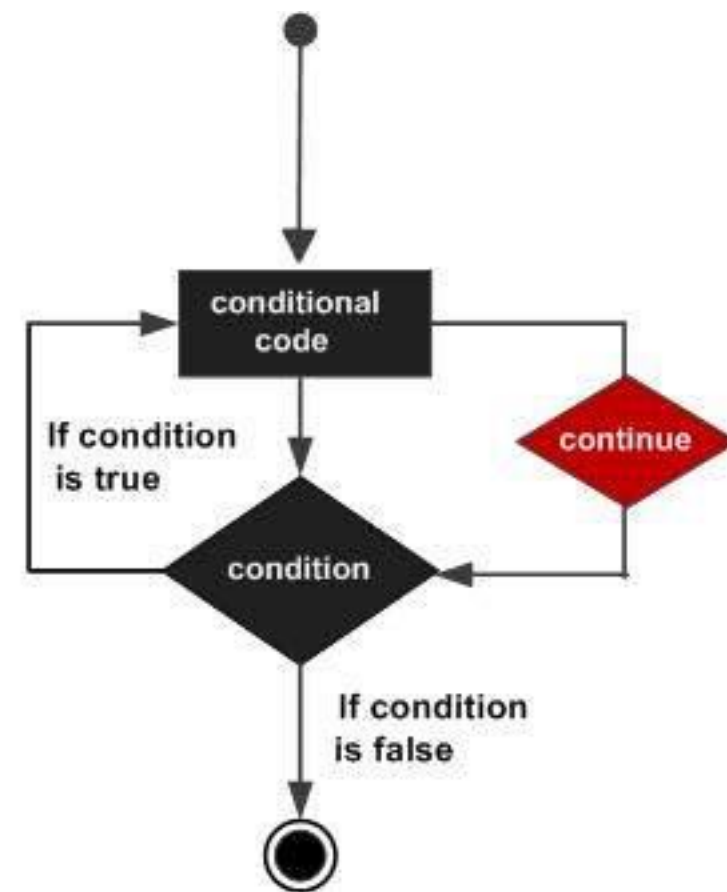


Python Basics: Loop Control Statements



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

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



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

Exercise # 5

- Given the list G (of dictionaries) below, do the following:
 - Write a for loop that prints out all names.
 - Write another for loop that adds up all ages and print the sum at the end.



Exercise # 6

- Given the list D (of integers) below, do the following:
 - Print only those values that are positive in the list



Recap with an Abstract View of a Computer

Processor (executes steps in program)

```
>>> welcomeString = "Hello World"
>>> eliteFloat = 1337 - 0.00001
>>> fullName = ["Mark", "Gottscho"]
>>> if fullName[1] == "Gottscho":
...     uni = "UCLA"
... else:
...     uni = "unknown"
...
>>> website = "www.google.com"
>>> print welcomeString
Hello World
>>> print "Searching on " + website + " is useful!"
Searching on www.google.com is useful!
>>> print str(eliteFloat) + " is close to 1337"
1336.99999 is close to 1337
>>> negativeInt = -28
>>> if negativeInt < 0:
...     print "< 0"
...
< 0
>>> my_numbers = [1, 2, 3, 4, 5]
>>> for i in range(0,5):
...     print my_numbers[i]
...
1
2
3
4
5
>>> negativeInt = -1
>>>
```

Memory (stores progress of program)

Location	Variable Name	Value
0	welcomeString	"Hello World"
1		
2	eliteFloat	1336.99999
3	my_numbers	[1, 2, 3, 4, 5]
4	fullName	["Mark", "Gottscho"]
5	uni	"UCLA"
6		-28
7	website	www.google.com
8	negativeInt	-1



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 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*.



Interacting with the External World

(Users, Files, Computers on the Internet ...)

- Printing to the Screen

```
>>> 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 opened file  
fo.close()
```

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

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



Interacting with the External World

(Users, Files, Computers on the Internet ...)

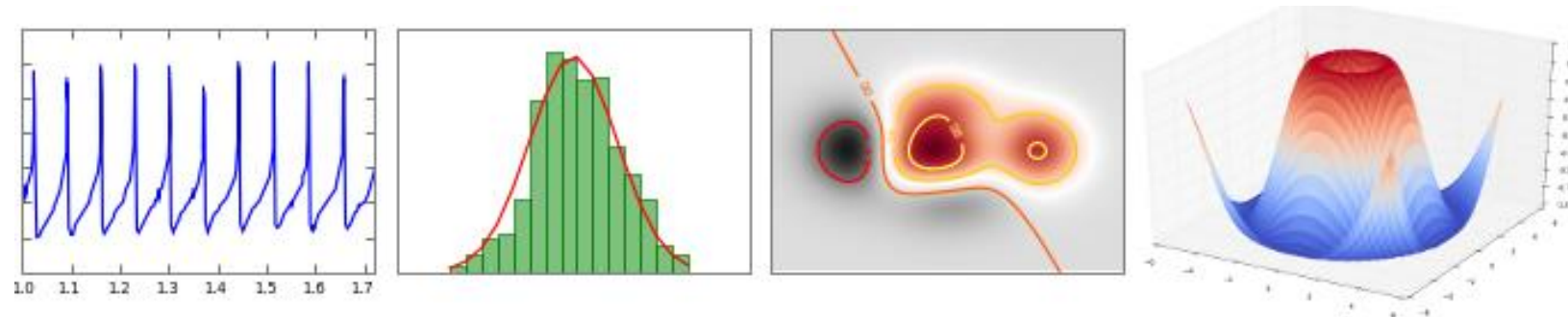
- Reading from Web Servers using url lib module

```
def wget(url):  
    try:  
        ufile = urllib.urlopen(url)  
        if ufile.info().gettype() == 'text/html':  
            print ufile.read()  
    except IOError:  
        print 'problem reading url:', url
```

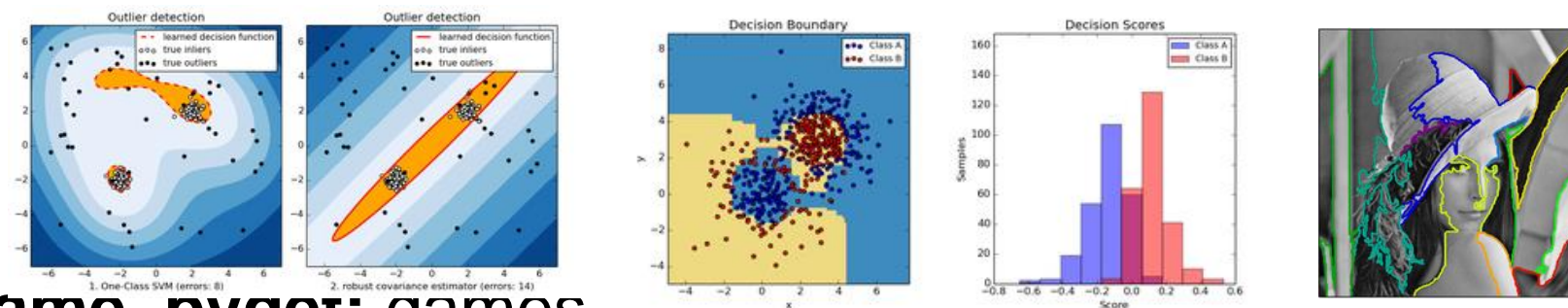


Some Very Useful Python Modules

- **matplotlib**: a python 2D plotting library



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



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



Assignment #1:

English Word to Pig Latin Translator

- Pig Latin is a language game, where you move the first letter of the word to the end and add "ay." So "Python" becomes "ythonpay."
- Write a program in **Assignment 1** section in jupyter notebook, which when run will prompt the user with the phrase "Enter a word:", and then output the word entered by the user translated into Pig Latin in lowercase unless the user's entry is empty, non-alphabetical, or has multiple word in which case the output should be "Error: incorrect input".
- Hint: break into your code into separate functions for getting user input, validating the input, translating it into Pig Latin etc.



Assignment #2:

Compute Statistics

- *Mean, mode, and median* are three common statistics of a set of numbers.
- Write a program in **Assignment 2** section in jupyter notebook, which when run will prompt the user with the phrase “Enter a comma separated list of numbers:”, and then output on three separate lines the mean, mode, and median of the entered numbers as “Mean = ...”, “Mode = ...” and “Median = ...”. The user may enter numbers as integers or real.
- Hint: Break into your code into separate functions for getting user input, computing mean, computing mode, computing median etc. Consider using the function *input()* instead of *raw_input()*- look up on the web as to how it works.



Assignment #3:

Frequency Analysis of Documents on the Web

- The relative frequencies (i.e. fractions of the total) of different letters in a document are very useful for a variety of purposes, such as compressing the document, encrypting it etc.
- Write a program in **Assignment 2** section in jupyter notebook, which when run will prompt the user with the phrase “Enter a URL:”, analyze the text at the URL, and then output a sorted (highest to lowest) listing of different English (a-z, A-Z) and numeric (0-9) characters and their respective fraction of the total one per line in the format “character fraction”. Count upper and lowercase versions of a character to be equivalent, and ignore any characters other than a-z, A-Z, 0-9. Print “Error reading URL” if the URL is not reachable or if it returns data of type other than something that starts with “text/”.
- Hint: what data type would you use?



Mini-Project: Networked Battleship Game

- You'll be given:
 - Network functions to talk to opponent
 - File **net.py** at <https://www.dropbox.com/s/buix5t7o5xjcsy3/net.py?dl=0>
- You must implement:
 - Ship placement
 - Hit detection
 - Game flow
 - In file **battleship.py** (to be submitted)
- Advanced assignment
 - Automated playing
 - AI player
 - Ship placement
 - Shooting strategy
- Game details: http://en.wikipedia.org/wiki/Battleship_%28game%29

	A	B	C	D	E	F	G	H	I	L
1										
2										
3										
4			X							
5						X	X			
6		X						X		X
7				X						X
8	X	X						X		
9										
10										



Battleship: General Implementation Strategy

- How to represent your (and the enemy's) board
- How to determine valid ship placement
- How to detect hits
- How to structure game flow
- Assume 10x10 board

