Introduction to Programming with Python – Basics

Swapnil Sayan Saha and Noor Nakhaei
PhD Students, ECE and CS
swapnilsayan@g.ucla.edu,
noornk@ucla.edu

Note: modified from original LACC slides by Dr. Lucas Wanner, Prof. Mani Srivastava, Dr. Mark Gottscho, Dr. Bharathan Balaji, Wojciech Romaszkan, Aishwarya Sivaraman, Lev Tauz, Ankur Sarker, Sandeep Singh Sandha and Shuyang Liu

Slides taken from UCLA-LACC Intro module

Introduction

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

The ***fundamental*** difference between a computer and a calculators 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.

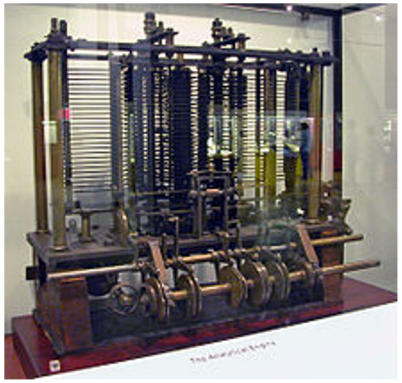
- A calculator can...
  - perform arithmetic operations
  - ~~perform logic operations~~
  - store values in memory
  - load values from memory

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

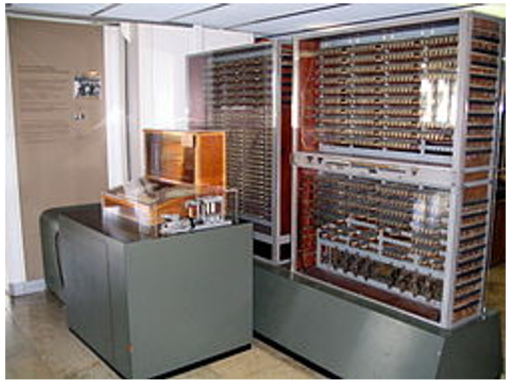
Unlike a calculator, a computer can make choices about what to do.
These are made using branches and loops in a procedure.
Essentially, decisions on what computation to do based on a condition.

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

History: Alan Turing

- British mathematician and a father of computer science: helped crack German encryption in WW2 using the Enigma computing machine

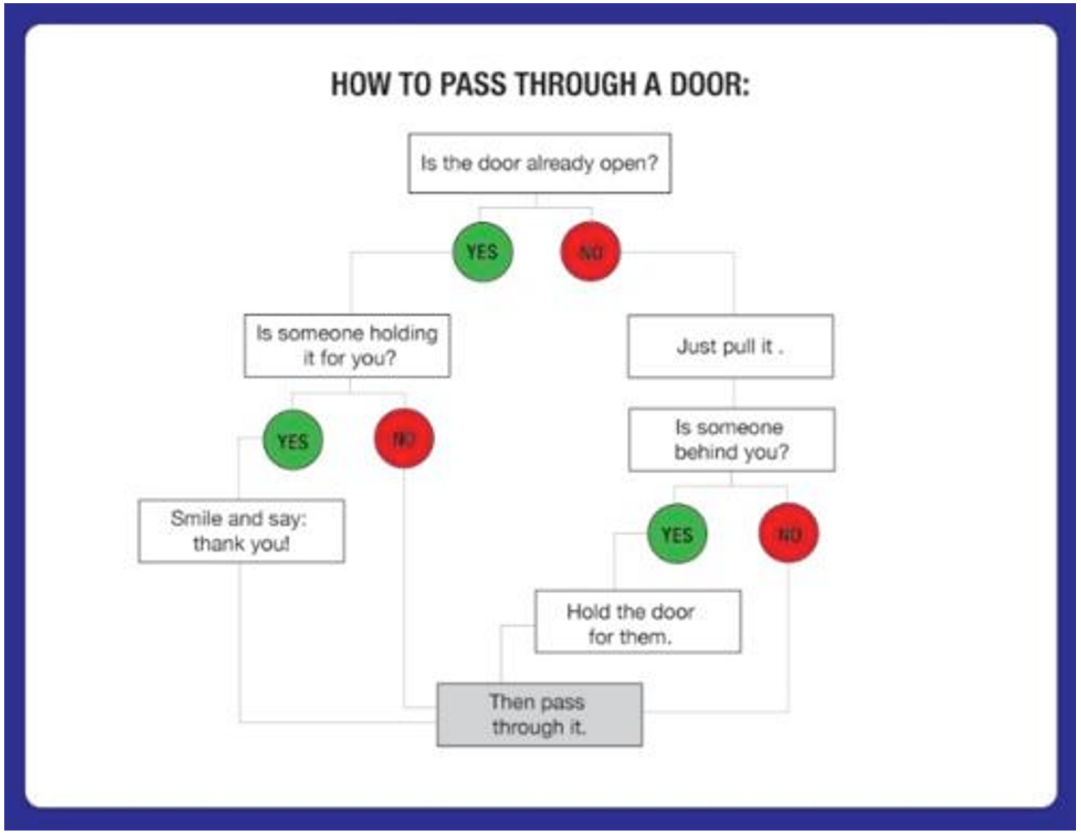
- Invented a hypothetical computing machine called the “Turing Machine” that helped formalize the theoretical limits of mechanical computation or algorithms, i.e. what sorts of things can be computed

- Universal Turing Machine == A Turing Machine that is able to simulate any other Turing Machine
  - Every real-world computer can be simulated by a UTM

- A real world computer or a programming language is called “Turing Complete” if it can simulate a UTM, or equivalently compute anything that can be computed

Conditional Branches

"A decision on which path to take based on some condition"



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

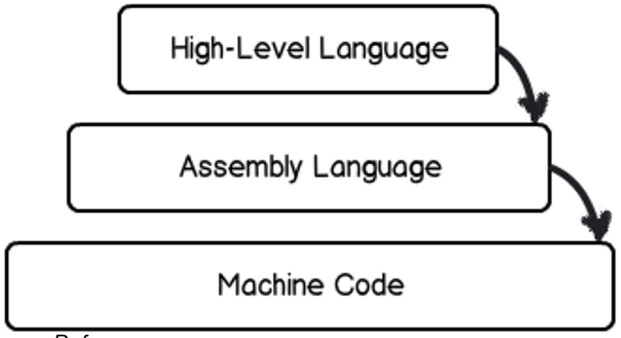
```
pass_door:
     ld register_1, door_status
     ld register_2, door_open
     beq register_1, register_2, holding
     call pull_door

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

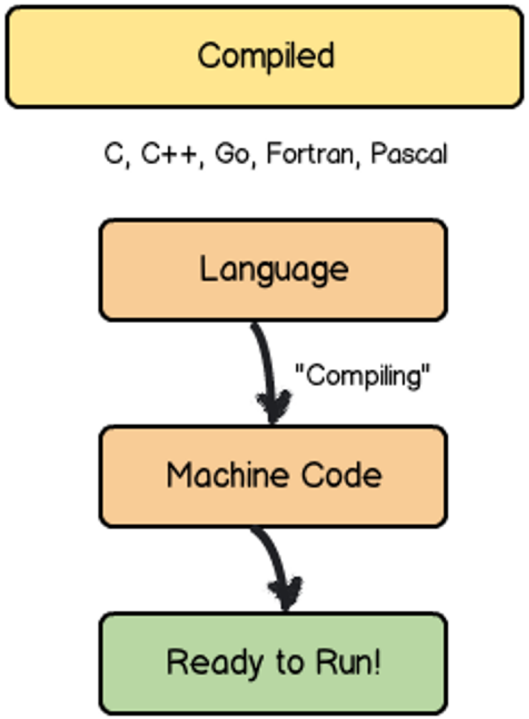
- Given a language providing…
  - function definitions and calls, f(x)
  - a library function, print("text")
  - comparisons, x == y
  - assignments, x = y
  - conditional statements, if (condition) … else …

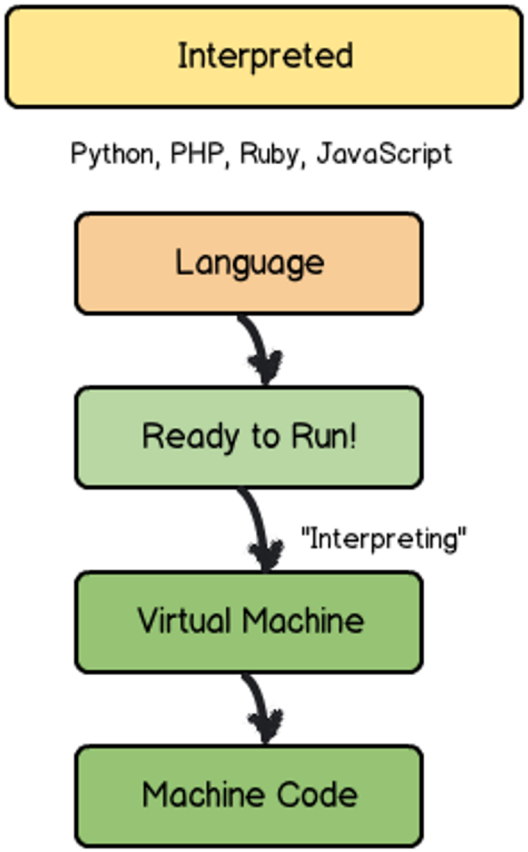


Ref:
http://shawnbiddle.com/js101/slides/class1.html#/6

- …how can we write the "pass door" program?

Compiled vs. Interpreted Programs





Ref:
http://shawnbiddle.com/js101/slides/class1.html#/10

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
```

Splicing

- We know that we can extract a single element from a list or string.
- Splicing is a type of indexing where we can get a chunk of a list or string.
- The syntax follows *name[start:end:step]* where the set of indices to extract are the same as if you got them from range.
- If only start is specified with a colon to the right, Python will splice the whole list upto that index. Similarly, if end is specified with a colon to the left, Python will splice the whole list starting at that index.

```
[1] L = [0,1,2,3,4,5,6]

    #Indexing
    a = L[3]
    print(a)
```

```
3
```

```
[5] a = L[2:5]
    print(a)
```

```
[2, 3, 4]
```

```
[6] a = L[2:6:2]
    print(a)
```

```
[2, 4]
```

```
[9] print(L[:4])
    print(L[2:])
```

```
[0, 1, 2, 3]
[2, 3, 4, 5, 6]
```

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:

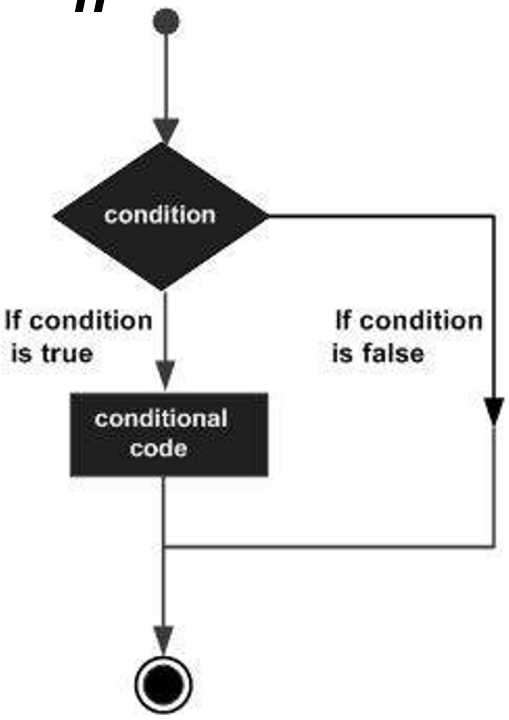
  - Add John's and Tom's surnames with "Surname" as a key. (You choose the surname)
  - Change John's city to NY.
  - Create a list F with both dictionaries as it's elements.
  - Print out the age of the first person on the list F.

  - Look at https://www.programiz.com/python-programming/methods/dictionary for various methods available for dictionary data type

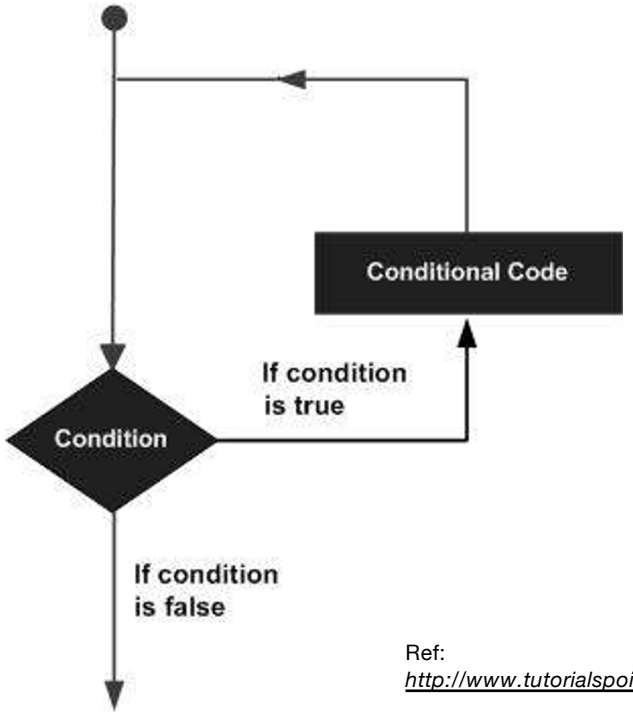
Beyond Sequential Execution: Control Flow Tools

- 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!

If…then



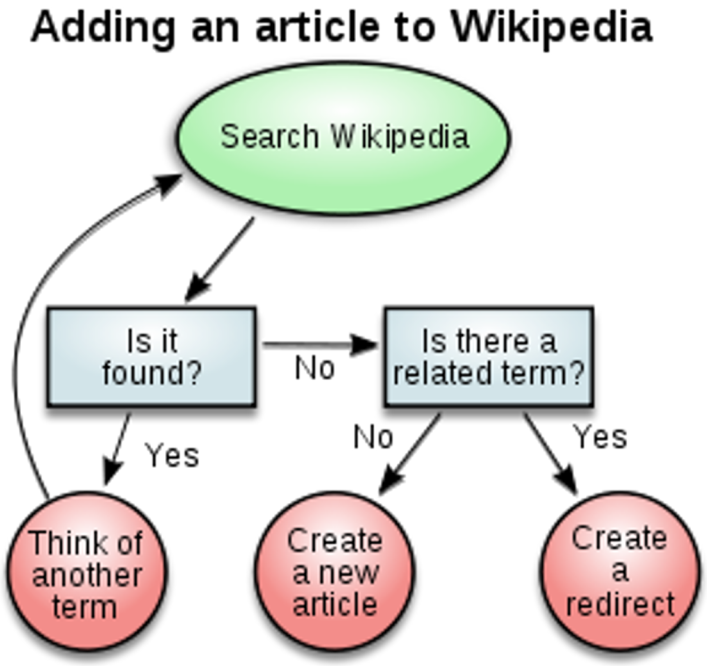
Loop



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:

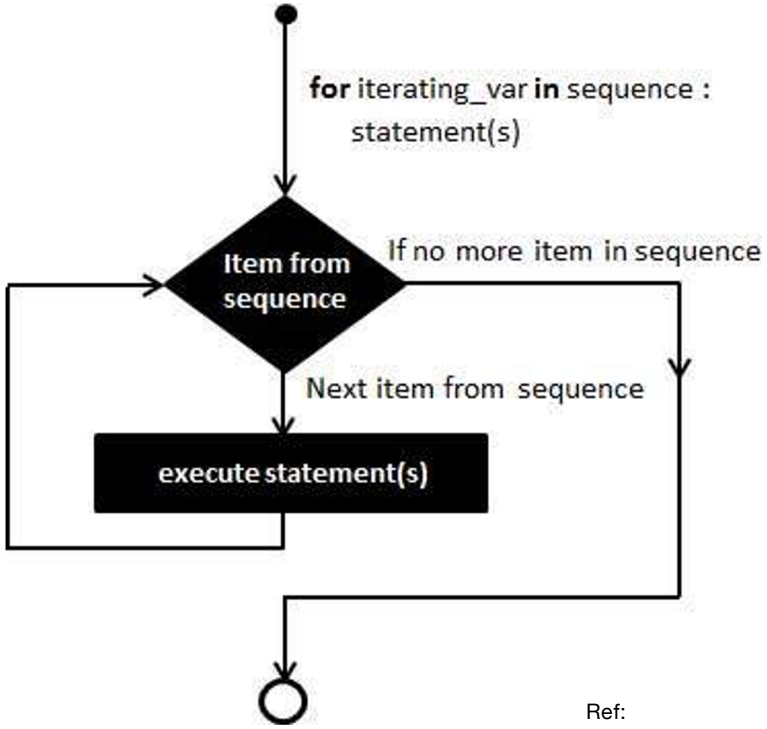
```
>>> x = int(raw_input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...      x = 0
...      print 'Negative changed to zero'
... elif x == 0:
...      print 'Zero'
... elif x == 1:
...      print 'Single'
... else:
...      print 'More'
...
More
```

- 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.

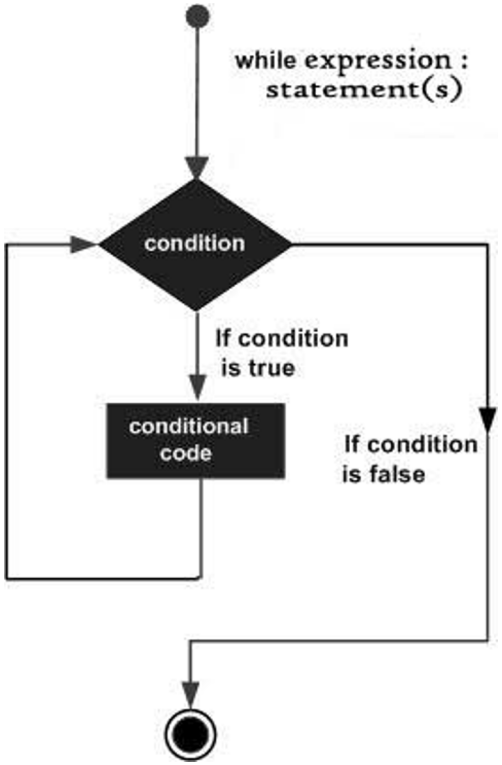
- Example:

```
>>> # 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

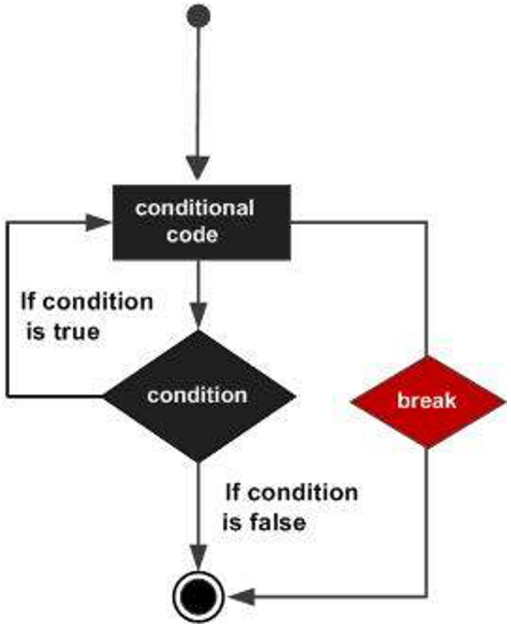
- Example:

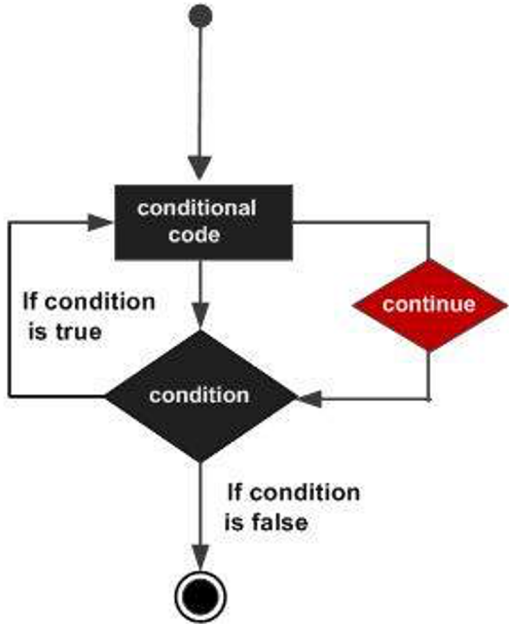
```
count = 0
while (count < 9):
   print 'The count is:', count
   count = count + 1

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/

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 # 4

 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 # 5

 Given the list D (of integers) below, do the following:

- Print only those values that are positive in the list

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 adds 5 onto the integer we passed into the function.

- The keyword return specifies the value return by the function. Functions do not have to return a value.

- Example:

```
In [58]: def add5(x):
             return x+5
```

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 # 6

- 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.

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)

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
```

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

Interacting with the External World *(Users, Files, Computers on the Internet …)*

Ref: *http://www.tutorialspoint.com/*

- 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: *input([prompt])*

```
# file.py

str = 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()
```

Interacting with the External World
(Users, Files, Computers on the Internet …)

- Reading from Web Servers using urllib module

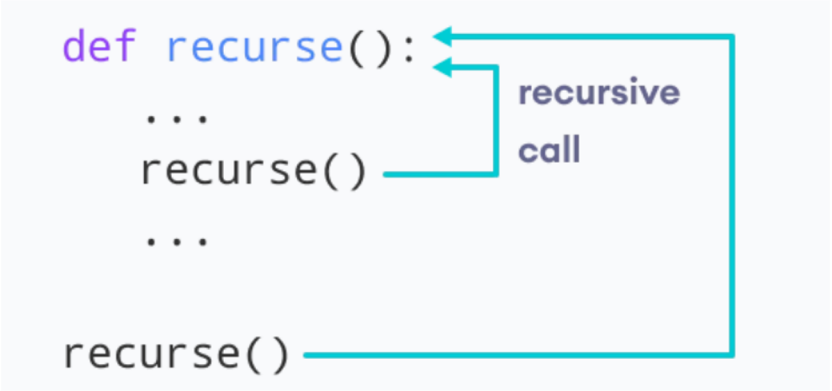
```
import urllib.request
def wget(url):
 try:
  with urllib.request.urlopen(url) as response
        html  = response.read().decode(response.headers.get_content_charset())
 except IOError:
  print('problem reading url:', url)
```

Exercise # 7

- Write code that does the following things:
  - Takes a raw input string and request for a number
  - Check if input string is a number
  - Convert input string to int
  - Use python's intrinsic math library to get the square root of the input number.

  - Look at https://docs.python.org/3/library/math.html for information on python's math library
  - Look at https://docs.python.org/3/library/stdtypes.html#string-methods for string functions

Recursion



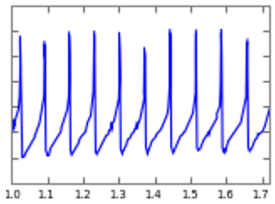
- A complex task can be broken down into simpler sub-tasks using recursion functions
- Recursion functions are expensive and hard to debug
- Example:
  - Fibonacci: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

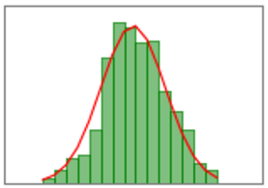
Recursion

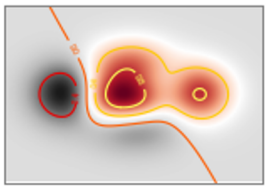
```
# Function for nth Fibonacci number

        def Fibonacci(n):
            if n<= 0:
                print("Incorrect input")
            # First Fibonacci number is 0
            elif n == 1:
                return 0
            # Second Fibonacci number is 1
            elif n == 2:
                return 1
            else:
                return Fibonacci(n-1)+Fibonacci(n-2)


        print(Fibonacci(0))
```

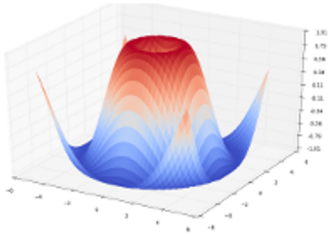
Some Very Useful Python Modules

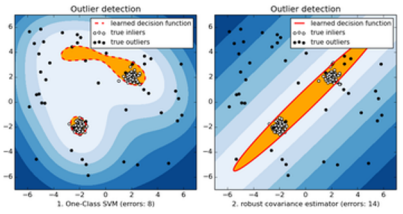
- **matplotlib:** a python 2D plotting library
- **NumPy:** computing with n-dimensional arrays
- **SciPy:** numerical integration and optimization
- **Sympy:** symbolic mathematics
- **scikit-learn:** data mining and machine learning
- **pygame, pyget:** games
- and many many more…

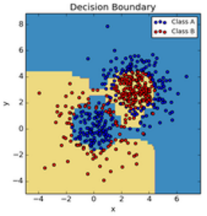


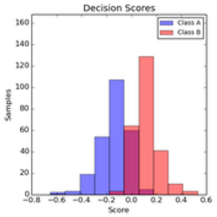


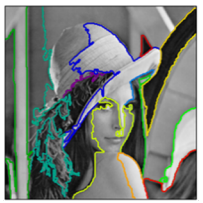












Assignment #1:
English Word to Pig Latin Translator

- 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: Tic-Tac-Toe

- Tic-tac-toe is a two player game played over a 3 by 3 grid where players taking turns making the spaces on the grid and the first play to place 3 of their marks in a diagonal, horizontal, or vertical row is the winner.

- Write a program in **Assignment 3** section in jupyter notebook, which will start a game of tic-tac-toe and print out the initial 3 by 3 board. It will then prompt a player to provide the location of their mark. By default, the first player will always put “X”. After every valid player input, the board is updated and printed. Make sure to check that a player’s move is valid otherwise the player has to make another input. This continues until either the game reaches a stalemate or one player is the winner.

- Hint: Which data type would you use to store the board?