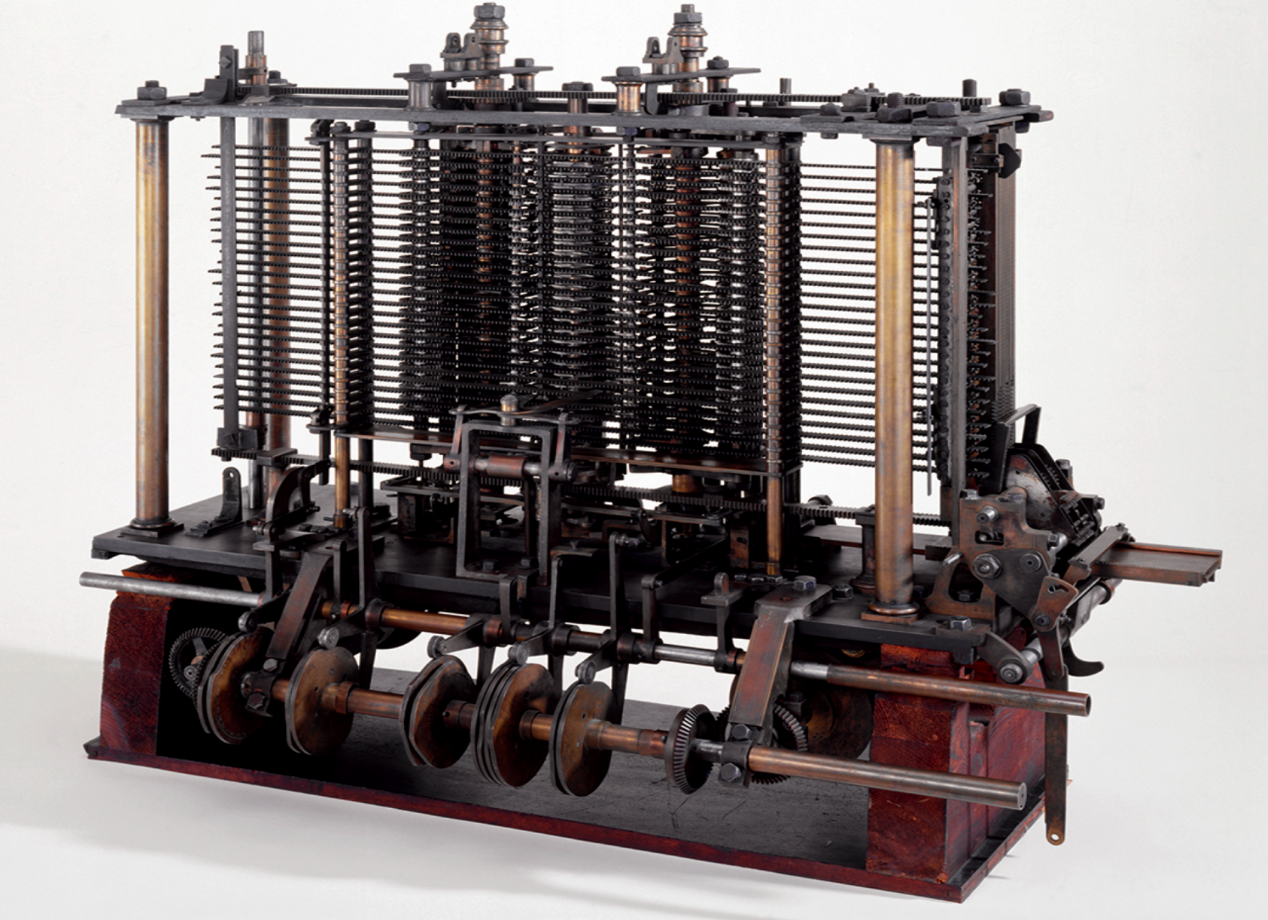
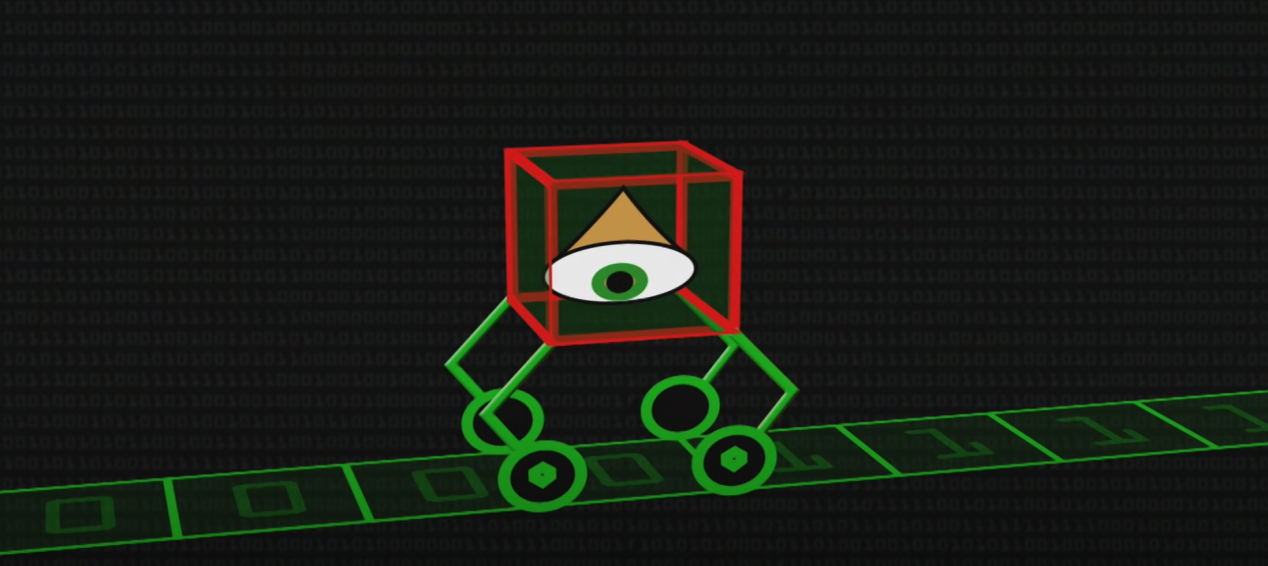
# Programming in Python

By Showmik Bose

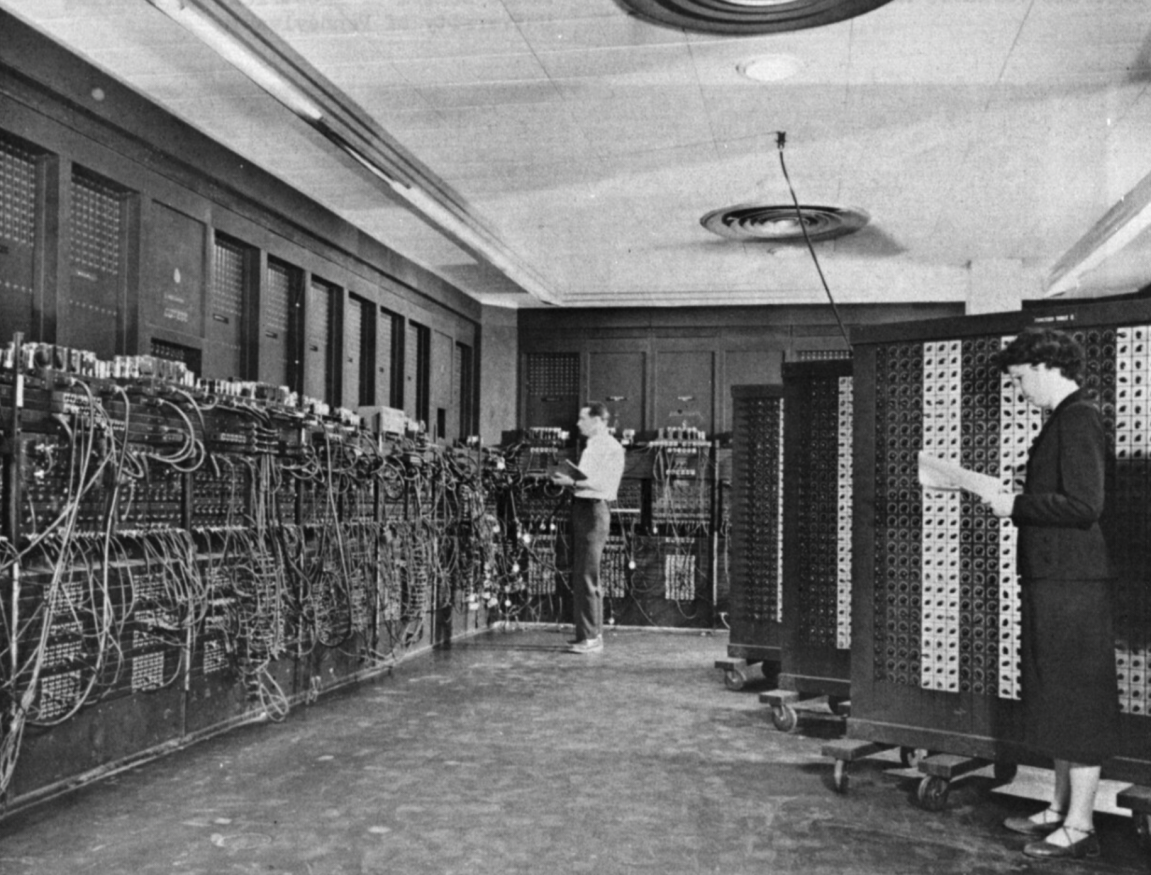
## Evolution of Computers



1837 : Babbage’s Analytical Engine



1936 : Alan Turing`



1946 : ENIAC

## Programming language evolution

1. 1958 - FORTRAN, ALGOL
2. 1959 - COBOL
3. 1972 - C and SQL
4. **FEBRUARY 20 1991 - PYTHON**

## Python Introduction

Created by Guido Van Rossum for :

* web development (server-side),
* software development,
* mathematics,
* system scripting.

## Interpreted vs Compiled Language

In a compiled language, the target machine directly translates the program. In an interpreted language, the source code is not directly translated by the target machine. Instead, a different program, aka the interpreter reads and executes the code.

Compiled Languages : C C++

Interpreted Languages : Python Ruby

* Advantages of Compiled Languages : Significantly faster
* Advantages of Interpreted Languages : More flexible

## Installing Python

Head over to : <https://www.python.org/downloads/>

For linux export PATH="$PATH:/path\_to\_python/python" and press Enter.

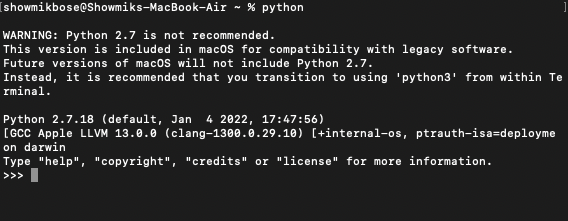
For windows add to Path variable.

## Python Basics

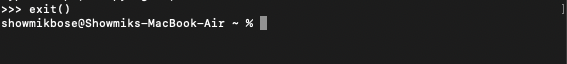
Open a terminal / Command Prompt and type

* $ python

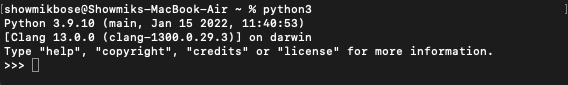
Note : The $ in front of codes symbolise the $ prompt and is not meant to be typed in. It could be % in mac or #.



* To exit python prompt

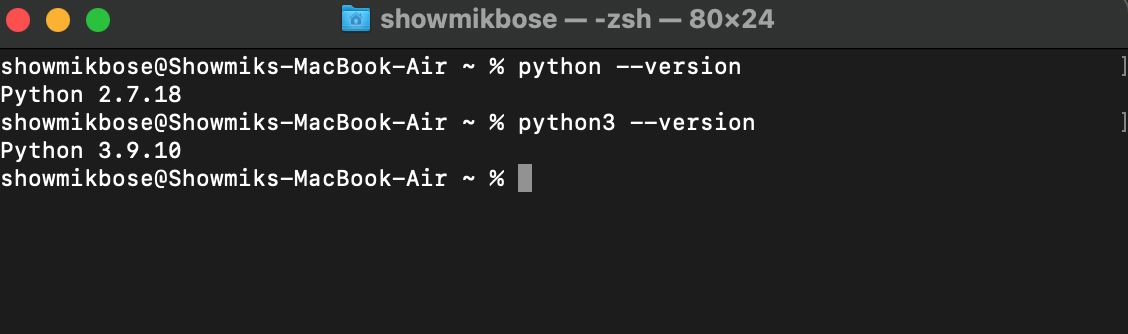


* If you have multiple versions of python installed then :

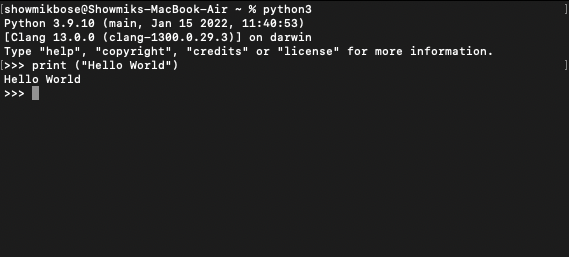


We will work with python3

* Let us check the Python Version



### Print Statement



## Python Identifiers

Identifier starts with a letter A to Z or a to z or an underscore (\_) followed by zero or more letters, underscores and digits (0 to 9).

Although the following words are reserved and hence can’t be used as identifiers :

|  |  |  |
| --- | --- | --- |
| and | exec | not |
| assert | finally | or |
| break | for | pass |
| class | from | print |
| continue | global | raise |
| def | if | return |
| del | import | try |
| elif | in | while |
| else | is | with |
| except | lambda | yield |

## Blocks in Python

Python unlike other languages doesn’t have specific blocks that define the start and end of code. Blocks of code are created by indentation.

Following is a correct block of code :

if True:

print ("True")

else:

print ("False")

But this one below is not correct :

if True:

print ("True")

else:

print ("False")

## Comments in Python

In Python there are a few different ways in which you can comment out lines.

#This is a comment

"""  
This is a

Multi line

Comment.  
"""

A comment means that the interpreter will ignore these lines and would not consider them as part of code.

## Variables in Python

A variable is just a placeholder. Whenever you create a variable you are simply reserving some space in the memory. In python you don’t need to explicitly specify the variable type. What does that mean? Well in some languages like Java and C, you need to specify whether the variable will hold an integer or a character or a list. But in Python you can create a variable without defining its type. Thus it will look like this in python

counter = 100 # An integer assignment

miles = 1000.0 # A floating point

name = "John" # A string

title = 'Rambo' # Another way of string declaration

It is a general convention to start the variable naming with small cases.

Multiple Assignments :

You can assign multiple variables in a single statement.

The value of a, b and c will be assigned to 1 in this case.

a = b = c = 1

Assigning multiple objects with multiple variables :

a,b,c = 1,2,"john"

**Type Casting :**

Casting is nothing but conversion of type. Say what ?!!

If we assign a variable as an integer but then we need to use it as a string that is when we use Type casting.

a = 3 # This is an integer variable

b = str(a) # b is a string holding the type casted value of a.

Let us try to find the variable type from the variable itself.

To do this, we will need to do the following :

a = 3 # This is an integer variable

print ( type(a) )

## Standard Data Types in Python

* Numbers
* Strings
* List
* Tuples
* Dictionary
* Numbers

Number data types store numeric values. Number objects are created when you assign a value to them:

var1 = 1

var2 = 10

* Strings

Strings in Python are identified as a contiguous set of characters represented in the quotation marks.

str= “Hello World”

Strings in python can be **sliced**.

The slice operator [:] allows us to take out slices out of a python string.

print (str) # Prints complete string

print (str[0]) # Prints first character of the string

print (str[2:5] ) # Prints characters starting from 3rd to 5th

print (str[2:] ) # Prints string starting from 3rd character

print (str \* 2 ) # Prints string two times

Concatenating Strings :

str = “hello”

str2 = “world”

print(str+str2)

Escape sequences :

We want the following part within quotes

 greatest glory in living lies not in never falling

str = “the greatest glory in living lies not in never falling, but in rising every time we fall”

print (str) # Prints complete string

But if we put quote in between the string above it’ll start throwing errors :

str = “the “greatest glory in living lies not in never falling”, but in rising every time we fall”

print (str) # Will throw error

Use escape sequence to make it work :

str = “the \“greatest glory in living lies not in never falling\”, but in rising every time we fall”

print (str) # Will not throw error

* List

List contains comma separated items

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]

As you see above, a list can contain multiple different type of items.

Here are some manipulations that can be done to the lists.

print (list) # Prints complete list

print (list[0]) # Prints first element of the list

print (list[1:3]) # Prints elements starting from 2nd till 3rd

print (list[2:] ) # Prints elements starting from 3rd element

* Tuple

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )

A tuple has Paranthesis () while a list has brackets []

The major difference between a list and a tuple is that tuples are immutable. Which means, you can’t change the contents of the tuple once you’ve created it.

Tuples can be manipulated the same way as list except that it can not be modified. This can be better realised in the example below :

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]

tuple[2] = 1000 # Invalid syntax with tuple

list[2] = 1000 # Valid syntax with list

* Dictionaries

Dictionary are key value pairs. They are enclosed by curly braces.

dict = {'name': 'john','code':6734, 'dept': 'sales'}

Dictionaries can also be manipulated

Print ( mydict ) # Prints complete dictionary

Print ( mydict.keys() ) # Prints all the keys

Print ( mydict.values() ) # Prints all the values

### Python Operators

Python language supports the following types of operators.

* Arithmetic Operators
* Comparison (Relational) Operators
* Assignment Operators
* Logical Operators
* Bitwise Operators
* Membership Operators
* Identity Operators
* Arithmetic Operators

Following are the arithmetic operators :

1. + Addition
2. - Subtraction
3. \* Multiplication
4. / Division
5. % Modulus
6. \*\* Exponential
7. // Floor division

* Comparison Operator

Following are the comparison operators:

1. == Is equals
2. != not equals
3. <> not equals
4. > greater than
5. < less than
6. >= greater than or equals
7. <= less than or equals

* Assignment Operator

1. =
2. +=
3. -=
4. \*=
5. /=
6. %=
7. \*\*=
8. //=

* Bitwise Operator

1. &
2. |
3. ^
4. ~
5. <<
6. >>

* Logical Operator

1. and
2. or
3. not

* Membership Operator

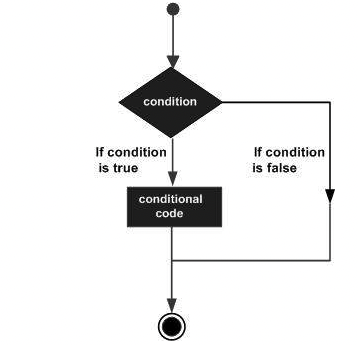
1. in
2. not in

* Identity Operator

1. is
2. is not

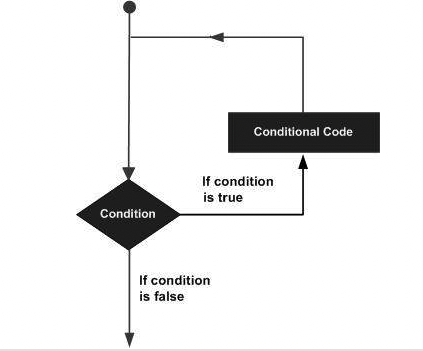
### Decision Making

Based upon whether a condition is evaluated to true or false we take a decision.



### Looping

Recursively evaluating a condition and performing a series of actions



There are two major types of loops in Python

1. **for** loop - To iterate over a sequence
2. **while** loop - To validate a condition first and then proceed to the loop

count = 0

while (count < 9):

print ('The count is:', count)

count = count + 1

for letter in **'Python':** # First Example

print (letter)

#Assigning a list and looping over it

fruits = ['banana', 'apple', 'mango']

for fruit in **fruits**:

print ('Current fruit :', fruit)

fruits = ['banana', 'apple', 'mango']

for index in range(len(fruits)):

print ('Current fruit :', fruits[index])

### Functions

Function is a reusable block of code that is used to perform a pre defined set of action.

Functions introduce reusability in code. To write a function, use the “def” keyword.

def functionname( parameters ):

"function\_docstring"

function\_suite

return [expression]

The above snippet describes the structure of a function.

### Classes and Objects

A class is a blueprint from which objects are created. Class has different properties, these properties become the characteristic of the objects.

* Classes are user defined data structures.
* Variables in class are called attributes of the class.
* Class attributes are always public and can be accessed by a “.”
* Class methods always have a “self” parameter in the method definition
* The \_\_init\_\_ method acts as a constructor to the class. If you want to initialize any value to the attributes whenever the class is instantiated, that’s when you need the init method.
* Instance variables are different from class variables and they define the characteristics of each object uniquely.

Sample Code Used in Lecture -

class Student:  
 # Class Attribute  
 school = "JJTech"  
  
 def \_\_init\_\_(self, student\_name, student\_class, age):  
 self.student\_name = student\_name  
 self.student\_class = student\_class  
 self.age = age  
  
 def get\_name(self):  
 return self.student\_name  
  
 def get\_school(self):  
 return self.school  
  
 def get\_age(self):  
 return self.age  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 hola = Student("Hola", "python", 10)  
 bose = Student("Showmik Bose", "terraform", 27)  
 enny = Student("Enny", "Docker",25)  
 print(enny.get\_age())  
 # print(bose.get\_name(), bose.get\_school())

### Boto3 and Automating the AWS environment

First install boto3 using

pip install boto3

Use the quickstart guide for your set up <https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html>

Sample Code Used in Lecture :

import boto3  
  
  
def init\_client(type):  
 client = boto3.client(type)  
 return client  
  
def describe\_volume(client):  
 response = client.describe\_volumes()  
 return response['Volumes']  
  
def del\_vol(client, vol\_id):  
 response = client.delete\_volume(VolumeId = vol\_id)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 type = "ec2"  
 client = init\_client(type)  
 vols = describe\_volume(client)  
 for vol in vols:  
 print(vol['VolumeId'])  
 del\_vol(client,vol['VolumeId'])  
 print(vol["VolumeId"], " deleted")

### Working with Lambdas

AWS lambda is a form of serverless compute. The code is executed based on the response of events in AWS services such as adding/removing files in S3 bucket, updating Amazon dynamo dB tables, HTTP request from Amazon API gateway etc.

To get working with AWS Lambda, we just have to push the code in AWS Lambda service. All other tasks and resources such as infrastructure, operating system, maintenance of server, code monitoring, logs and security is taken care by AWS.

Advantages of Lambdas

* Only worry about the code
* Monitoring is already taken care of
* Easy alterting integration
* Billing based on use
* Multi Language support

import boto3  
  
  
def del\_inst(client, list\_of\_inst\_ids):  
 response = client.terminate\_instances(  
 InstanceIds= list\_of\_inst\_ids  
 )  
  
def desc\_inst(client):  
 response = client.describe\_instances(  
 Filters=[  
 {  
 'Name': 'tag:Name',  
 'Values': [  
 'To\_Be\_Deleted',  
 ]  
 },  
 ]  
 )  
 return response  
  
def lambda\_handler(event, context):  
 list\_of\_inst\_ids = []  
 ec2\_client = boto3.client("ec2")  
 all\_instances = desc\_inst(ec2\_client)  
 # print (all\_instances['Reservations'])  
 for reservation in all\_instances['Reservations']:  
 for instance in reservation['Instances']:  
 # create\_img(ec2\_client, instance['InstanceId'], 'eks-management-instance-boto-backup-2')  
 list\_of\_inst\_ids.append(instance['InstanceId'])  
 print("Instances To Be Deleted : ",list\_of\_inst\_ids)  
 del\_inst(ec2\_client, list\_of\_inst\_ids)  
 print("Instances Deleted")

### Coderpad Interview :

Input : Sample Access Log

192.168.1.2 - - [17/Sep/2013:22:18:19 -0700] "GET /abc HTTP/1.1" 404 201

192.168.1.1 - - [17/Sep/2013:22:18:19 -0700] "GET /favicon.ico HTTP/1.1" 200 1406

192.168.1.2 - - [17/Sep/2013:22:18:27 -0700] "GET /wp/ HTTP/1.1" 200 5325

192.168.1.2 - - [17/Sep/2013:22:18:27 -0700] "GET /wp/wp-content/themes/twentytwelve/style.css?ver=3.5.1 HTTP/1.1" 200 35292

192.168.1.3 - - [17/Sep/2013:22:18:27 -0700] "GET /wp/wp-content/themes/twentytwelve/js/navigation.js?ver=1.0 HTTP/1.1" 200 863

Parse the above log and print the IP That occurs the most.

*'''  
: Sample Access Log  
192.168.1.2 - - [17/Sep/2013:22:18:19 -0700] "GET /abc HTTP/1.1" 404 201  
192.168.1.1 - - [17/Sep/2013:22:18:19 -0700] "GET /favicon.ico HTTP/1.1" 200 1406  
192.168.1.2 - - [17/Sep/2013:22:18:27 -0700] "GETp/ /w HTTP/1.1" 200 5325  
192.168.1.2 - - [17/Sep/2013:22:18:27 -0700] "GET /wp/wp-content/themes/twentytwelve/style.css?ver=3.5.1 HTTP/1.1" 200 35292  
192.168.1.3 - - [17/Sep/2013:22:18:27 -0700] "GET /wp/wp-content/themes/twentytwelve/js/navigation.js?ver=1.0 HTTP/1.1" 200 863  
'''*inp = '''192.168.1.2 - - [17/Sep/2013:22:18:19 -0700] "GET /abc HTTP/1.1" 404 201  
192.168.1.1 - - [17/Sep/2013:22:18:19 -0700] "GET /favicon.ico HTTP/1.1" 200 1406  
192.168.1.2 - - [17/Sep/2013:22:18:27 -0700] "GET /wp/ HTTP/1.1" 200 5325  
192.168.1.2 - - [17/Sep/2013:22:18:27 -0700] "GET /wp/wp-content/themes/twentytwelve/style.css?ver=3.5.1 HTTP/1.1" 200 35292  
192.168.1.3 - - [17/Sep/2013:22:18:27 -0700] "GET /wp/wp-content/themes/twentytwelve/js/navigation.js?ver=1.0 HTTP/1.1" 200 863'''  
counter = {}  
for line in inp.split("\n"):  
 # print(line)  
 item = line.split(" ")  
  
 if item[0] in counter:  
 counter[item[0]]+=1  
 else:  
 counter[item[0]]=1  
print(counter)  
highest\_value = max(counter, key=counter.get)  
print(highest\_value)