CCNA-DevNet (200-901) v1.0

Introduction

IP ROUTE
TECHNOLOGIES PVT LTD

Trainer Profile

- 21 years of experience in IT industry focussed on automation
- 4 years of experience in corporate training
- Areas of expertise include Python, Ansible, REST APIs, Automation, AI/ML, DevOps tools
- Extensive experience in building integrations using APIs on Cisco ACI, Meraki, DNAC
- Experience in building end to end custom integration solutions using ServiceNow, Grafana, Splunk, Slack, Microsoft Teams and other tools

Ravinuthala Nagaraj

Cisco Certified DevNet Specialist - Core

IP ROUTE TECHNOLOGIES PVT LTD

Things to note…

- Be regular to the sessions
- Where possible, practice along with the trainer
- Take homework seriously and try to complete
- Do not hesitate to ask questions
- Any general topics, raise in WhatsApp group or approach support teams

IP ROUTE
TECHNOLOGIES PVT LTD

Topics

- Python
- Ansible
- REST APIs
- GIT

Introduction to Python

- What is Cisco DevNet?
  - Cisco’s developer program
  - To help developers and IT professionals
  - To write applications and integrations with Cisco products, platforms and APIs
  - Kind of bridge between software engineering and networking
  - Bringing best of both worlds together
  - The things that we will be doing will require a “development environment”

What is a development environment?

- Collection of software, tools and resources to help us do our job
- Can create scripts, write programs, automate tasks, build integrations
- All these need tools setup on system and known as “development environment”
- Can be classified as local, hosted and cloud based
- Local – setup everything on our own machine
- Hosted – setup on a VM by a hosting provider, used by us
- Cloud Based – setup on one a machine hosted by one of the cloud providers like AWS, Azure, Google Cloud etc.

Development Environment (Cont.)

- Collection of software, tools and resources to help us do our job
- Can create scripts, write programs, automate tasks, build integrations
- All these need tools setup on system and known as "development environment"
- Can be classified as local, hosted and cloud based
- Local – setup everything on our own machine
- Hosted – setup on a VM by a hosting provider, used by us
- Cloud Based – setup on one a machine hosted by one of the cloud providers like AWS, Azure, Google Cloud etc.

Development Environment (Cont.)

- Development environment is typically a combination of:
  - Shells / Command lines (bash, cmd, Terminal)
  - Source control systems
  - Programming languages
  - Operating systems (Linux, Mac, Windows)

Scripting vs Programming

Scripting languages are:

- Low-level
- Not general purpose
- Ideal for automating simple tasks
- Not modular or reusable
- Usually interpreted

Programming languages

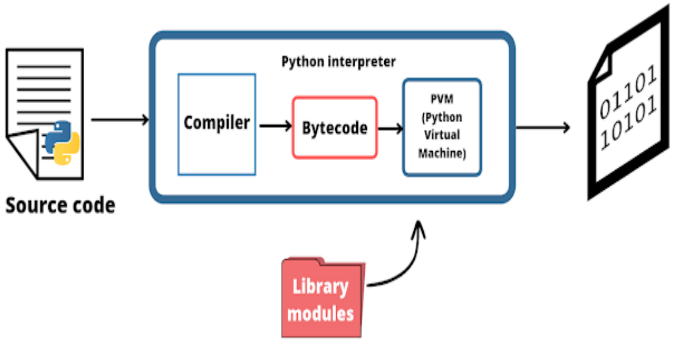
- Can be low-level or high-level
- General purpose
- Can be used to write complex applications
- Support modular programming and are reusable
- Can be compiled or interpreted

Fundamentals of Programming

- A program is a set of instructions given to a computer to perform a specific operation using some data
- When the program is executed, raw data is processed into a desired output format
- These programs are written in high-level programming languages which are close to human languages
- They are then converted to machine understandable low-level languages and executed

What is Python?

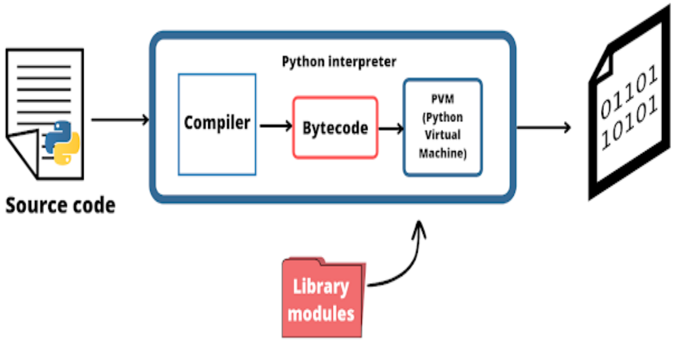
- It is programming language created by "Guido Van Rossum" in 1991.
- It is a general purpose & High-Level programming Language.
- Python is commonly used for developing websites and software, task automation, data analysis, and data visualization etc...



IP ROUTE TECHNOLOGIES PVT LTD

What is Python?

- It is programming language created by "Guido Van Rossum" in 1991.
- It is a general purpose & **High-Level programming** Language
- Python is commonly used for developing websites and softv task automation, data analysis, and data visualization etc...



- **** High-Level programming: --->**

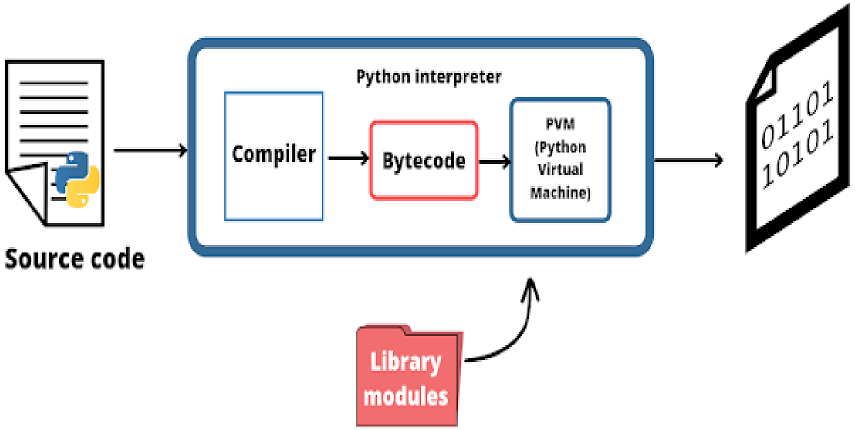
- It is closer to Humans. i.e., human readable form.
When you write a High-level programming code,
it is not directly compiled on machine (CPU) but gets interpreted.
- Which means that it needs to run by another program. This program is known as “

3. What is Python?

- It is programming language created by "Guido Van Rossum" in 1991.
- It is a general purpose & **High-Level programming** Language.
- Python is commonly used for developing websites and software, task automation, data analysis, and data visualization etc...

- **** High-Level programming: --->**

- It is closer to Humans. i.e., human readable form.
  When you write a High-level programming code,
  it is not directly compiled on machine (CPU) but gets interpreted.
- Which means that it needs to run by another program. This program is known as **Interpreter".**
  - **"JAVA"** is an exception. It is both compiled and interpreted.)



- **Python Interpreter**

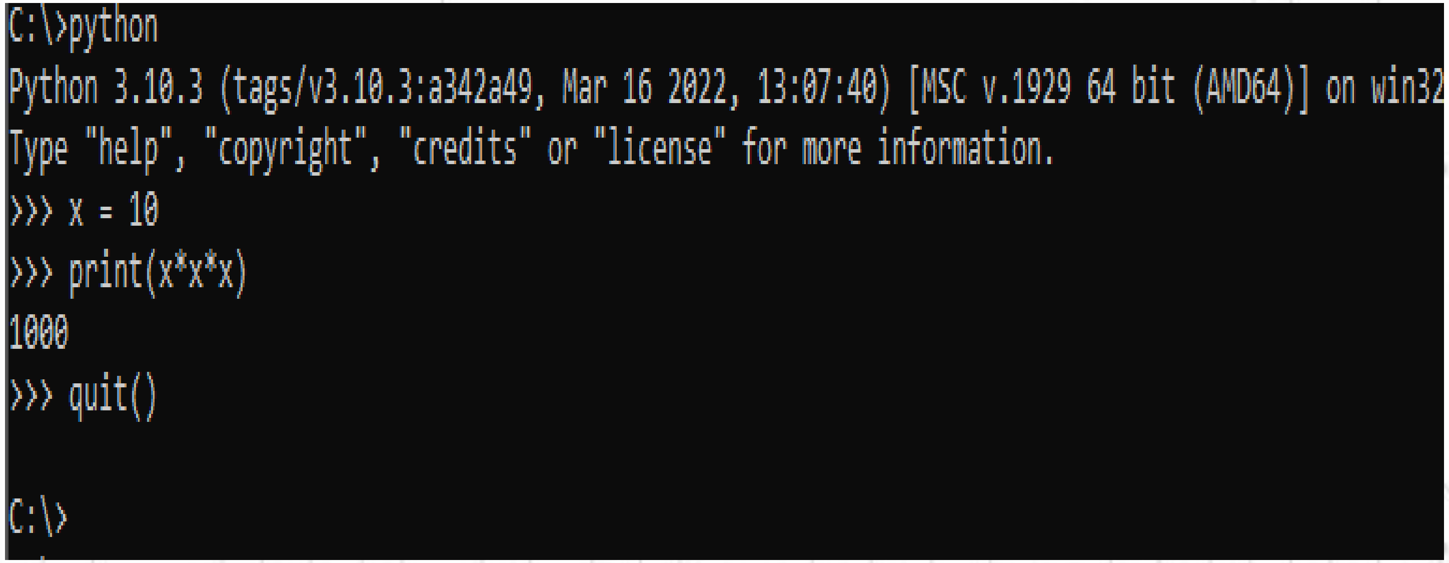
- An Interpreter is a program that converts the code a developer writes into an intermediate language, called the byte code.
- It converts the code line by line, one at a time and translates till the end. (Stops at the line where an error occurs, if any)
- The Python Interpreter, stored in the memory as a collection of instructions in binary form.

Internal working of python:

- ✓ The program gets compiled by the python compiler and checks for errors, if the compiler finds an error it throws an error message to the console.
- ✓ If there is no error, and the source code is well-formatted, the compiler converts the source code to Bytecode.
- ✓ The bytecode is then processed inside the Python Virtual Machine (PVM) and is being interpreted to give the actual machine code.
- ✓ The machine code is then executed by the CPU to return the output.

Python Modes:

Interactive mode: It is a command line shell, which gives immediate feedback for each statement, while running previous fed statement in active memory.



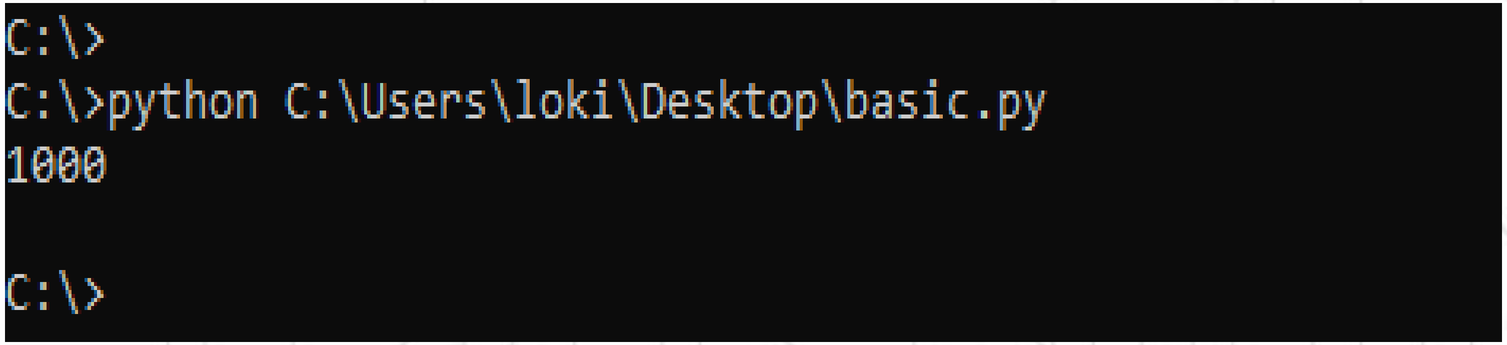
```
C:\>python
Python 3.10.3 (tags/v3.10.3:a342a49, Mar 16 2022, 13:07:40) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 10
>>> print(x*x*x)
1000
>>> quit()

C:\>
```

Script mode:

- For longer python codes/scripts; the script is written in a text file and saved as PY-script with an extension of **".py".**

- After writing and saving the code, the file is executed in **CMD prompt**.

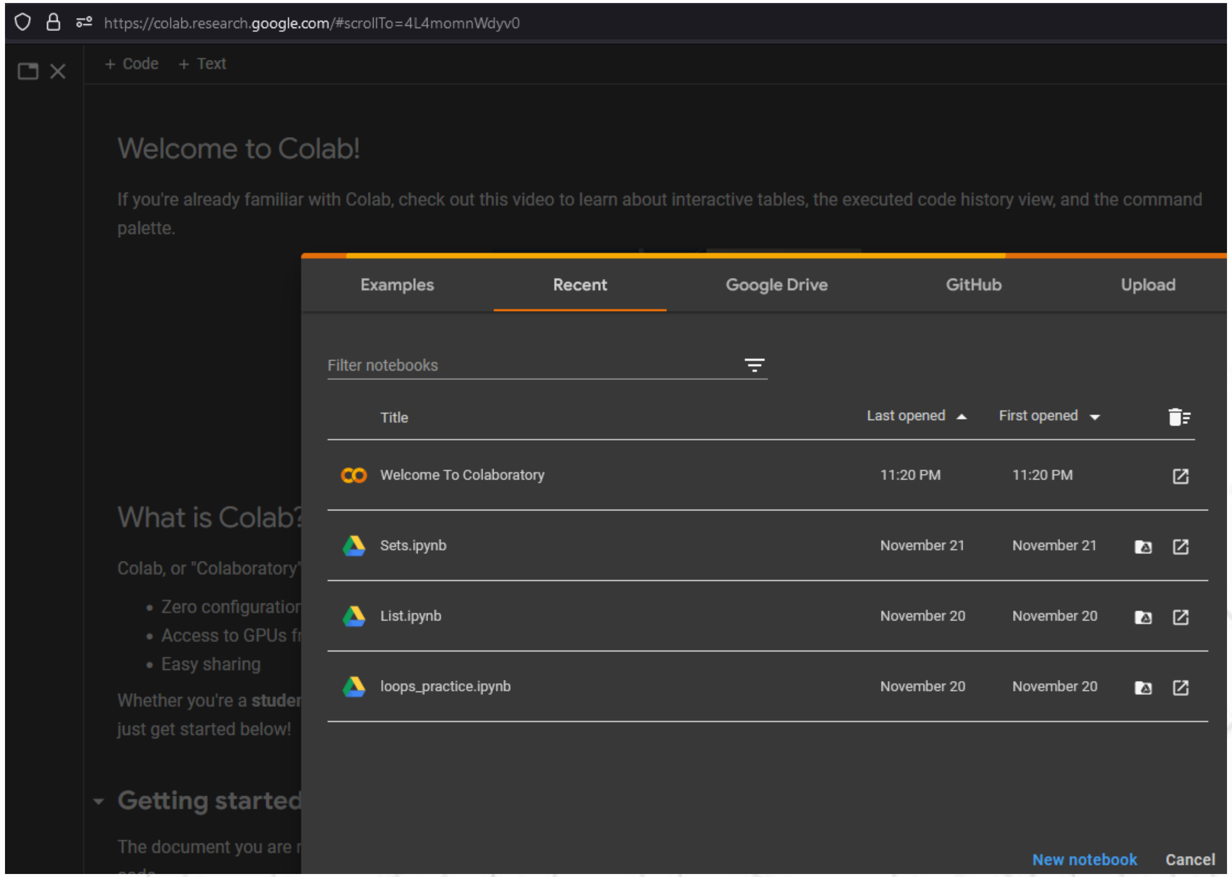
- You only need a Gmail account. We will use ▯ https://colab.research.google.com to learn the basics.



```
C:\>
C:\>python C:\Users\loki\Desktop\basic.py
1000

C:\>
```

Click on “New notebook”

A new page will open, rename the notebook with **“.ipynb”** extension



https://colab.research.google.com/#scrollTo=4L4momnWdyv0

Welcome to Colab!

If you're already familiar with Colab, check out this video to learn about interactive tables, the executed code history view, and the command palette.

What is Colab?

Colab, or "Colaboratory"

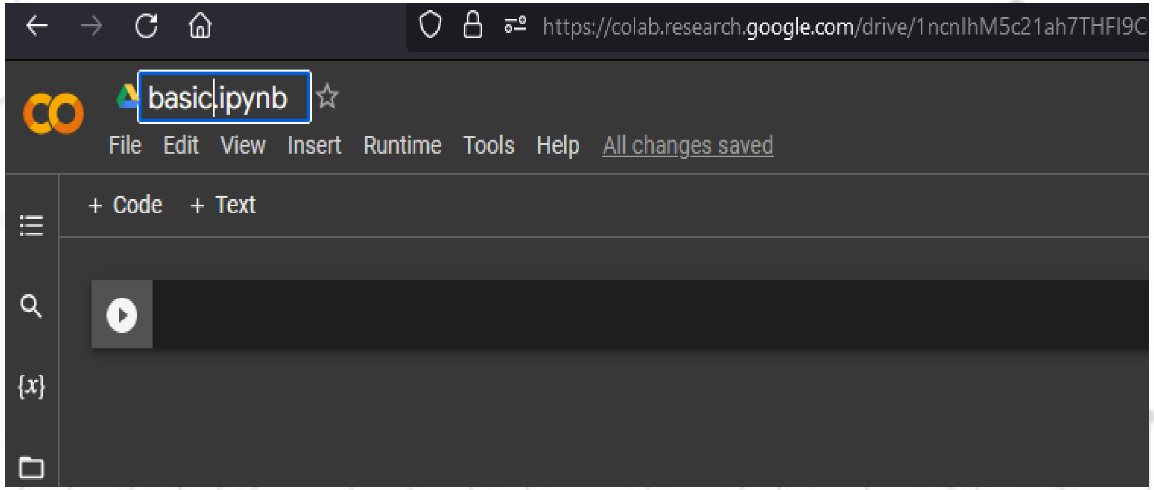
- Zero configuration
- Access to GPUs f
- Easy sharing

Whether you're a studer just get started below!

Getting started

The document you are r

Title	Last opened	First opened
Welcome To Colaboratory	11:20 PM	11:20 PM
Sets.ipynb	November 21	November 21
List.ipynb	November 20	November 20
loops_practice.ipynb	November 20	November 20



https://colab.research.google.com/drive/1ncnlhM5c21ah7THFI9C

basic.ipynb

All changes saved

What are Python Identifiers?

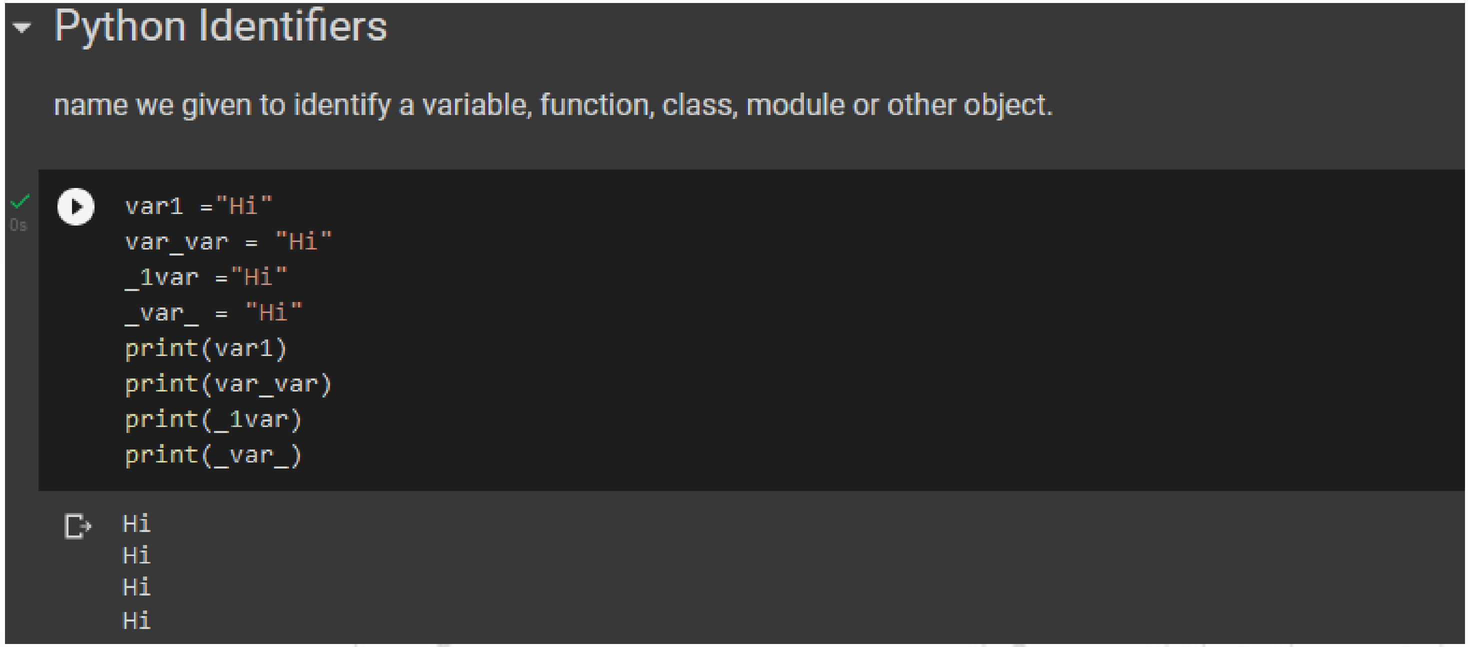
- Python Identifier is the name given to identify a variable, function, class, module or other object.
- Sometimes variable and identifier are often misunderstood as same but **they are not**.

✓ A variable is a memory location where a value can be stored.

✓ An identifier is the name given memory location where the variable is stored.

-

- **Rules for identifier names** —>> has to start with an alphabet **A to Z** or **a to z** or
- **an underscore (_)** followed by zero or more letters or underscores or digits. For example:



Python Identifiers

name we given to identify a variable, function, class, module or other object.

```
var1 ="Hi"
var_var = "Hi"
_1var ="Hi"
_var_ = "Hi"
print(var1)
print(var_var)
print(_1var)
print(_var_)
```

```
Hi
Hi
Hi
Hi
```

- An Identifier cannot start with digit. So, while var1 is valid, 1var is not valid.
- We can’t use special symbols like **!, #,@,%,$** etc in our Identifier.
- Identifier can be of any length.

A} Python Variables:

- Variable is containers which store values. A variable is created the moment we first assign a value to it.
- A Python variable is a name given to a memory location. It is the basic unit of storage in a program.
- So, Variables in Python are reserved memory locations.
-

- **Rules for creating variables in Python:**

✓ A variable name must start with a letter or the underscore character.

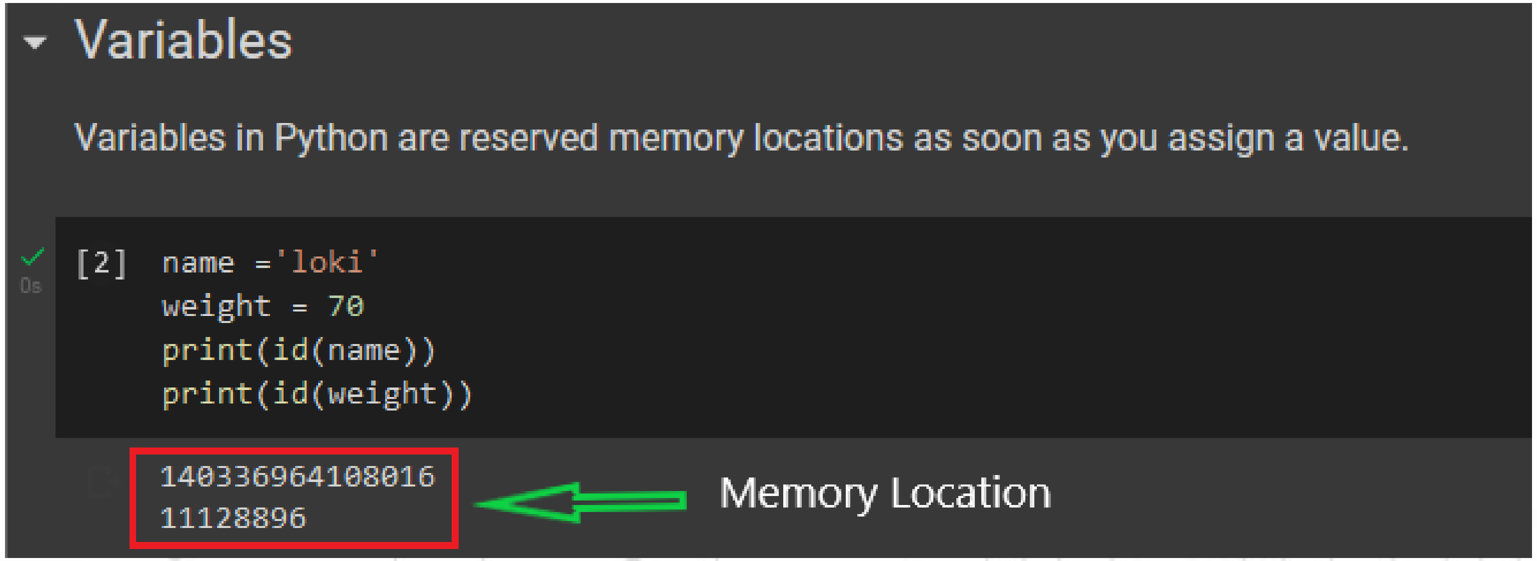
✓ A variable name cannot start with a number.

✓ A variable name can only contain alpha-numeric characters and **(A-z, 0-9, and _).**

✓ Variable names are case-sensitive (name, Name and NAME are three different variables).

✓ The reserved words(keywords) cannot be used naming the variable.

*Example**



Variables

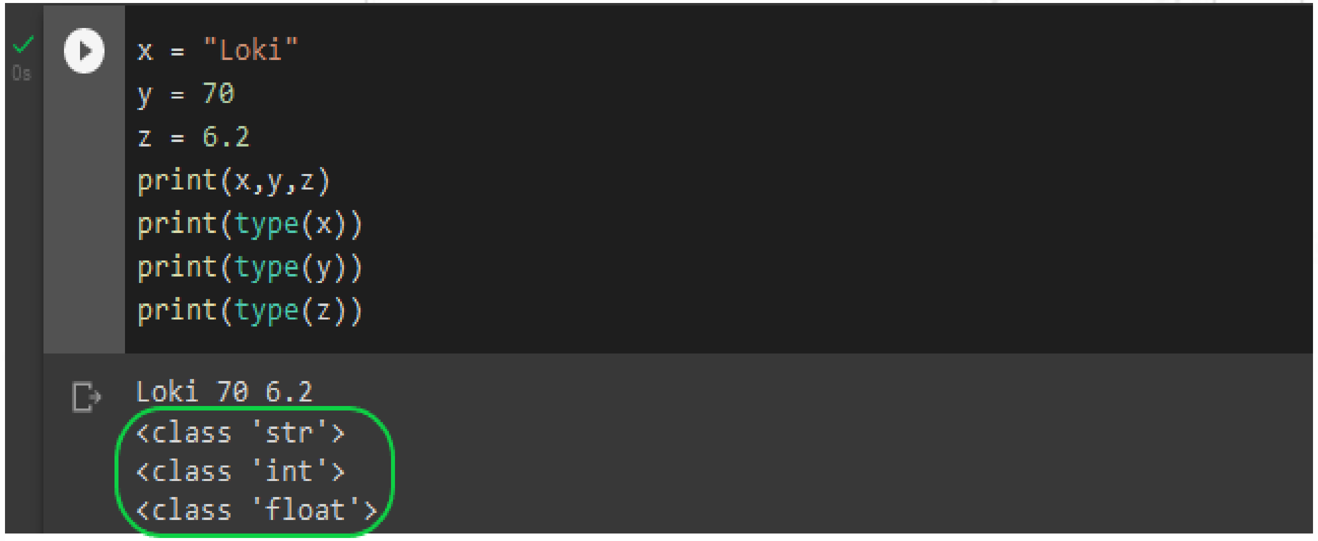
Variables in Python are reserved memory locations as soon as you assign a value.

```
[2]  name ='loki'
     weight = 70
     print(id(name))
     print(id(weight))
```

```
140336964108016
11128896
```

Memory Location

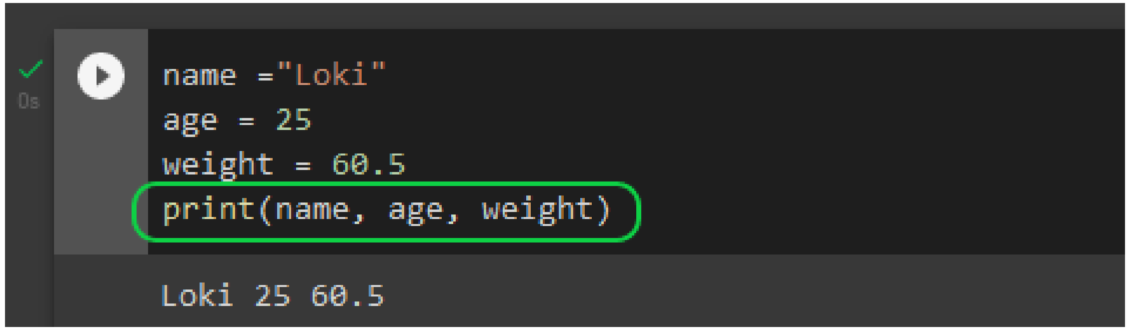
- A variable can be a **“String”, “Integer” & “Float”**
- **Integer:** Numeric values.
- **Float:** Variables that are intended to hold floating precession values.
- **String:** Variables that are intended to hold a string of letters.
- **Example’s***



```
x = "Loki"
y = 70
z = 6.2
print(x,y,z)
print(type(x))
print(type(y))
print(type(z))
```

```
Loki 70 6.2
<class 'str'>
<class 'int'>
<class 'float'>
```

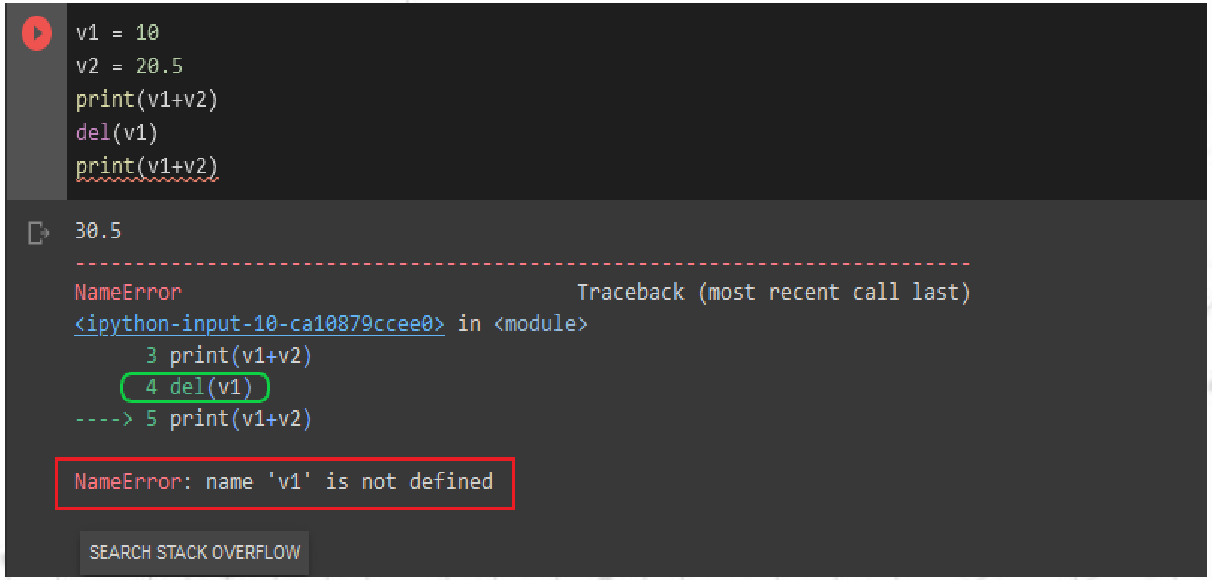
1)Assigning multiple variables at once:



```
name ="Loki"
age = 25
weight = 60.5
print(name, age, weight)
```

```
Loki 25 60.5
```

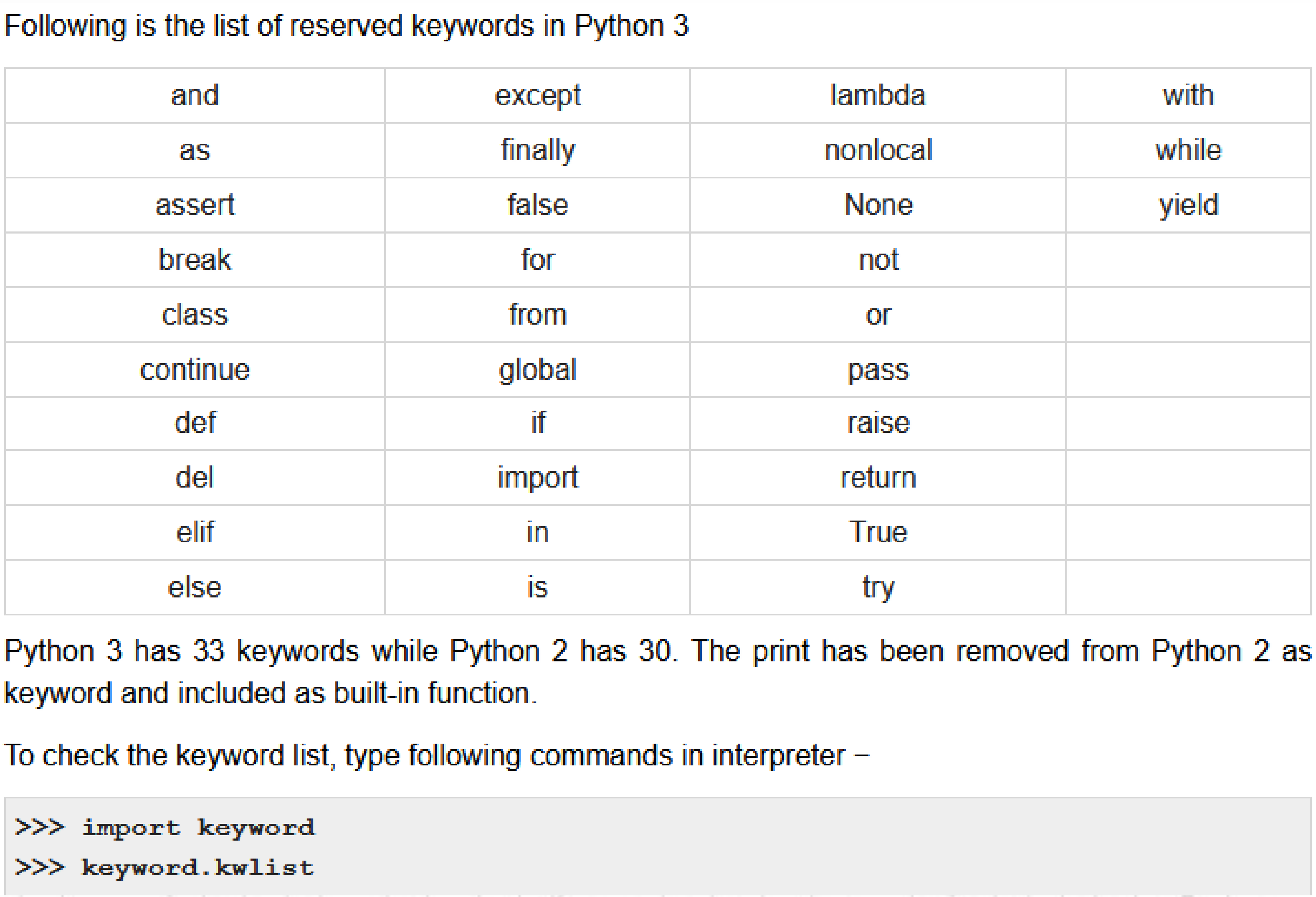
2) Deleting variables:



```
v1 = 10
v2 = 20.5
print(v1+v2)
del(v1)
print(v1+v2)
```

```
30.5
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-10-ca10879ccee0> in <module>
      3 print(v1+v2)
      4 del(v1)
----> 5 print(v1+v2)

NameError: name 'v1' is not defined
```

Reserved Key Words in Python:



Following is the list of reserved keywords in Python 3

and	except	lambda	with
as	finally	nonlocal	while
assert	false	None	yield
break	for	not	
class	from	or	
continue	global	pass	
def	if	raise	
del	import	return	
elif	in	True	
else	is	try	

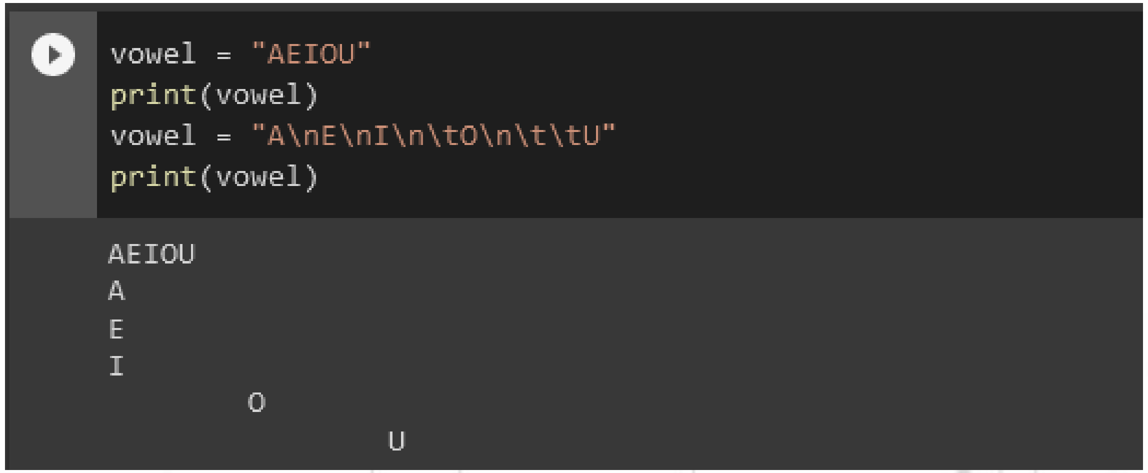
Python 3 has 33 keywords while Python 2 has 30. The print has been removed from Python 2 as keyword and included as built-in function.

To check the keyword list, type following commands in interpreter –

```
>>> import keyword
>>> keyword.kwlist
```

Multi-line statements:

- You can split a statement into multiple lines, if needed, as follows:



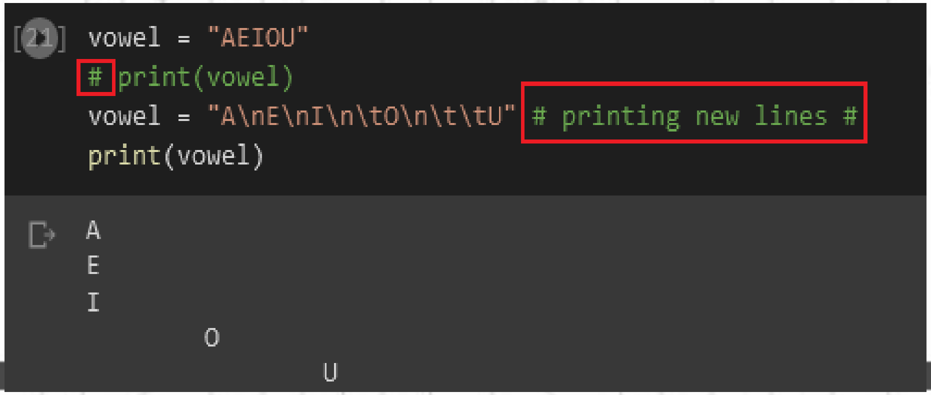
```
vowel = "AEIOU"
print(vowel)
vowel = "A\nE\nI\n\tO\n\t\tU"
print(vowel)
```

```
AEIOU
A
E
I
        O
                U
```

- **Comments:**

- Comments are used in any language to put some text for our reference without being executed.

- In Python **comments** start with **# (hash)** symbol



```
vowel = "AEIOU"
# print(vowel)
vowel = "A\nE\nI\n\tO\n\t\tU" # printing new lines #
print(vowel)
```

```
A
E
I
        O
                U
```

Data Types used in Python:

- As the name suggests DataType defines the type of the data stored in them.
- Depending on the datatype, the operations that can be performed and the storage mechanism varies.

- **Standard DataTypes are:**

A. String

B. Number

C. List

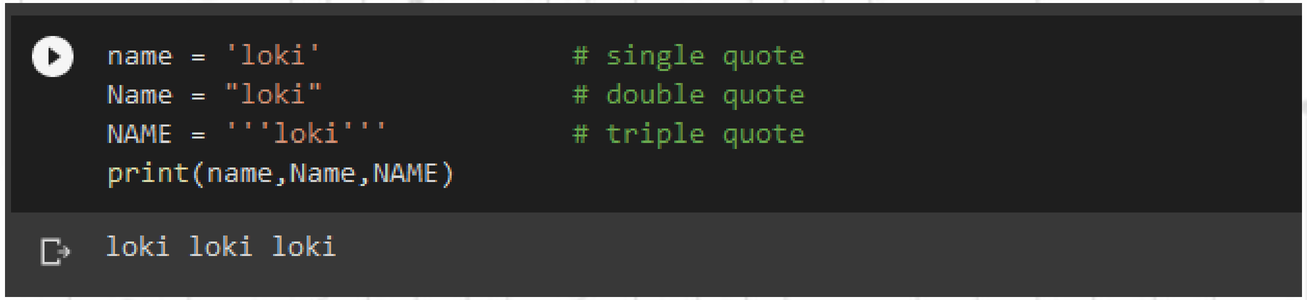
D. Tuple

E. Dictionary

- Let us see in details what each of them.

A. String:

- ✓ String in Python or any language for that matter are a contiguous set of characters represented in quotes.
- ✓ It can be single, double or triple quotes.
- ✓ So, in a way string in Python becomes an array or a list of characters
- ✓ Characters in String can be accessed using the **slicing operators [ ]** or **[:]**
- ✓ String index starts from **0** and also have **-ve** indexing.
- ✓ Strings are Immutable



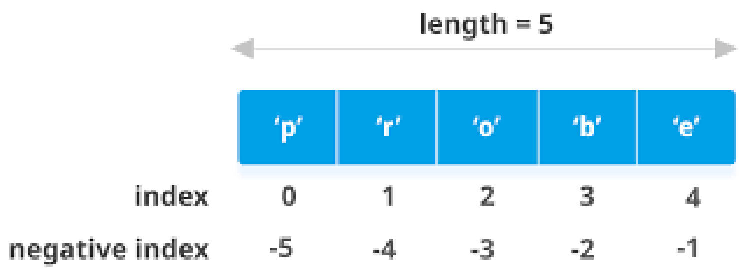
```
name = 'loki'              # single quote
Name = "loki"              # double quote
NAME = '''loki'''          # triple quote
print(name,Name,NAME)
```

```
loki loki loki
```

Use dir(variable) to find the lists of operations, you can perform on 'variable'

Some of the Basic Operations you can perform on a string.

- **Indexing:**

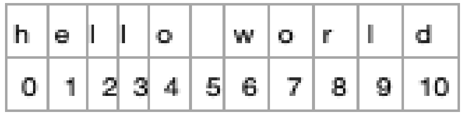


length = 5

	'p'	'r'	'o'	'b'	'e'
index	0	1	2	3	4
negative index	-5	-4	-3	-2	-1

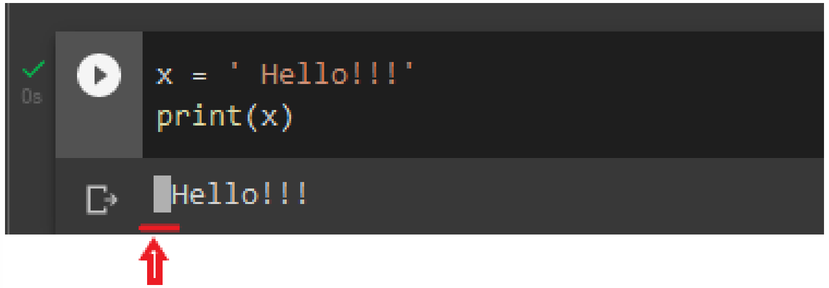
Example:

Table 1



h	e	l	l	o		w	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10

□ **Remove spaces from a string: (space between " " & character):**

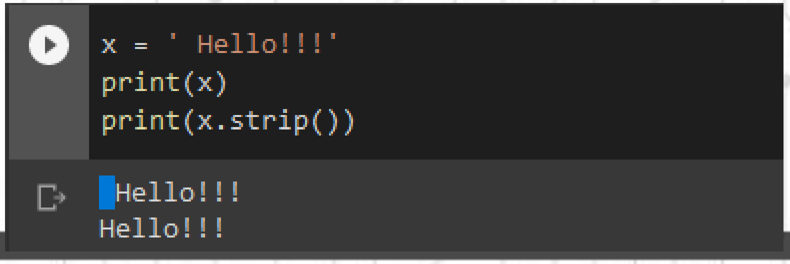


```
x = ' Hello!!!'
print(x)
```

```
 Hello!!!
```

space

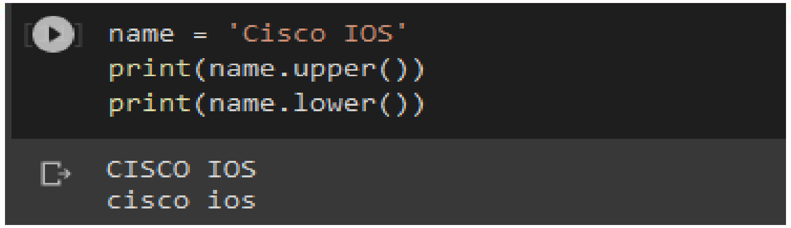
Remove the space:



```
x = ' Hello!!!'
print(x)
print(x.strip())
```

```
 Hello!!!
Hello!!!
```

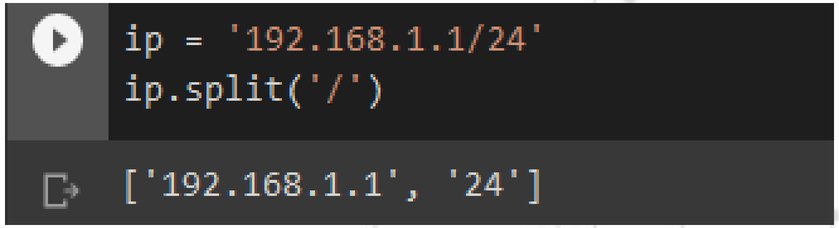
- **Lower and Upper case:**



```
name = 'Cisco IOS'
print(name.upper())
print(name.lower())
```

```
CISCO IOS
cisco ios
```

□ **Splitting a "String":**

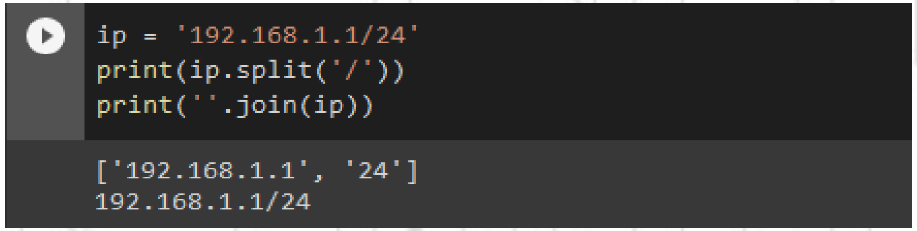


```
ip = '192.168.1.1/24'
ip.split('/')
```

```
['192.168.1.1', '24']
```

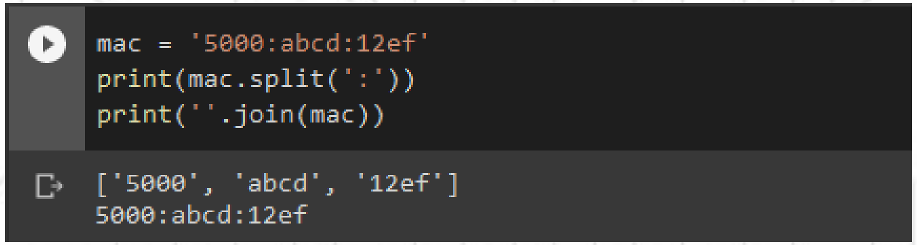
When you split a string, it is converted into Python List

□ **Joining back a "String":**



```
ip = '192.168.1.1/24'
print(ip.split('/'))
print(''.join(ip))
```

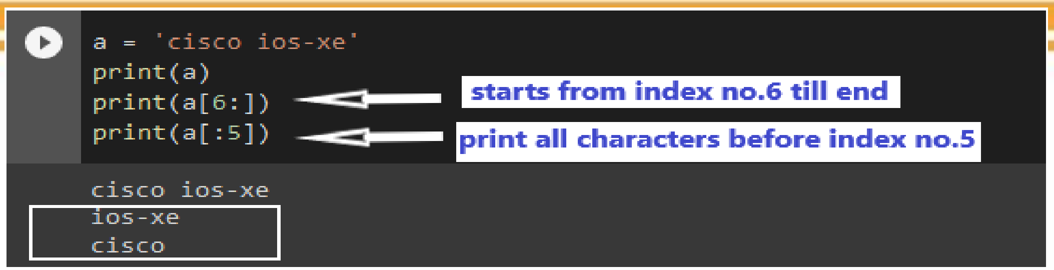
```
['192.168.1.1', '24']
192.168.1.1/24
```



```
mac = '5000:abcd:12ef'
print(mac.split(':'))
print(''.join(mac))
```

```
['5000', 'abcd', '12ef']
5000:abcd:12ef
```

- **Slicing Operator & Index in a "String":**



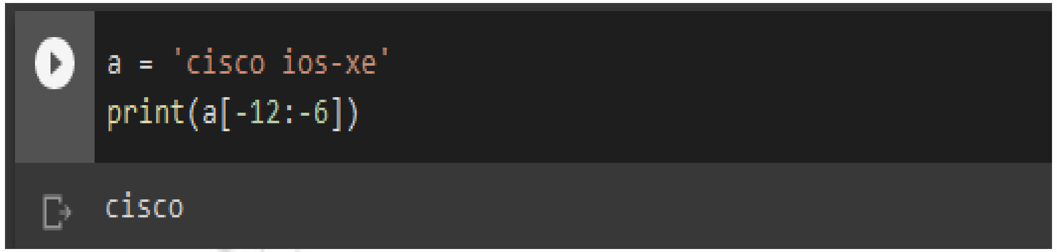
```
a = 'cisco ios-xe'
print(a)
print(a[6:])
print(a[:5])
```

starts from index no.6 till end

print all characters before index no.5

```
cisco ios-xe
ios-xe
cisco
```

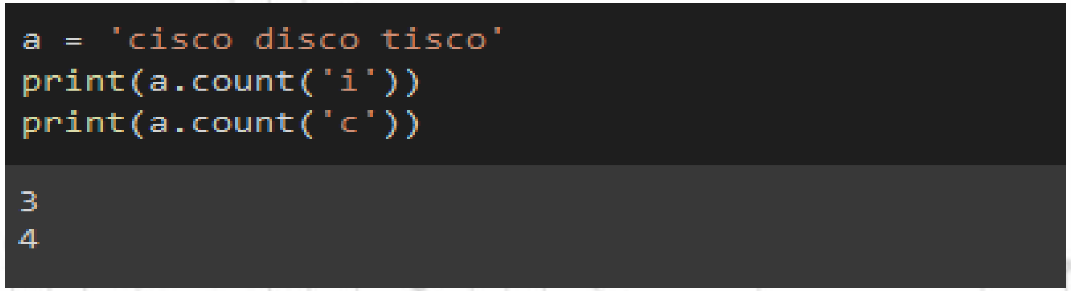
- **-ve Indexing:**



```
a = 'cisco ios-xe'
print(a[-12:-6])
```

```
cisco
```

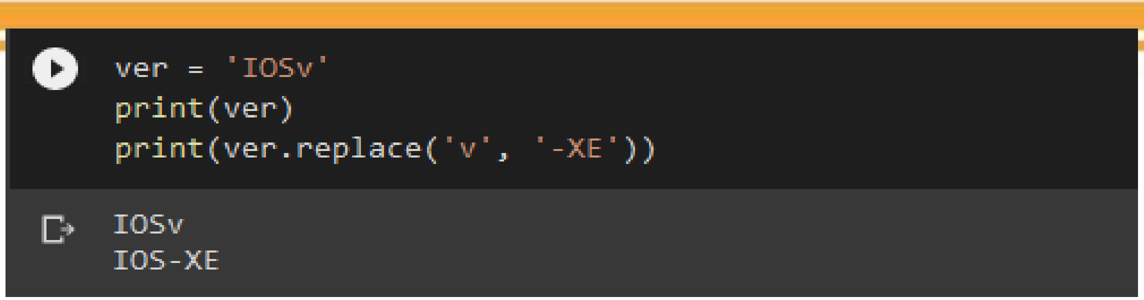
- **Get No. of characters in a "String" or Count:**



```
a = 'cisco disco tisco'
print(a.count('i'))
print(a.count('c'))
```

```
3
4
```

- **Replace a Character in a String**

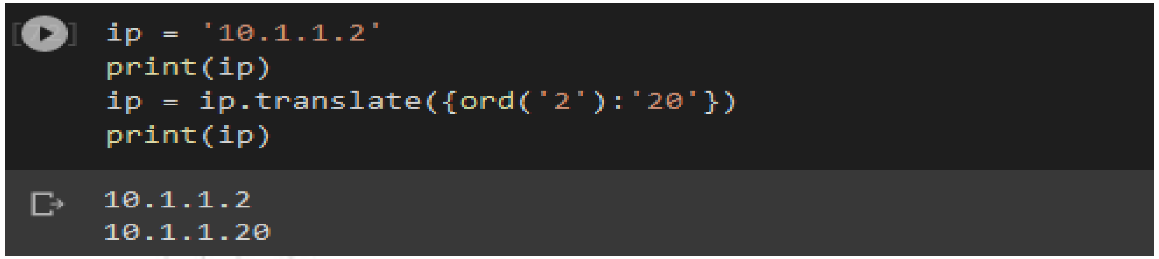


```
ver = 'IOSv'
print(ver)
print(ver.replace('v', '-XE'))
```

```
IOSv
IOS-XE
```

- **Or you can use ‘translate’**

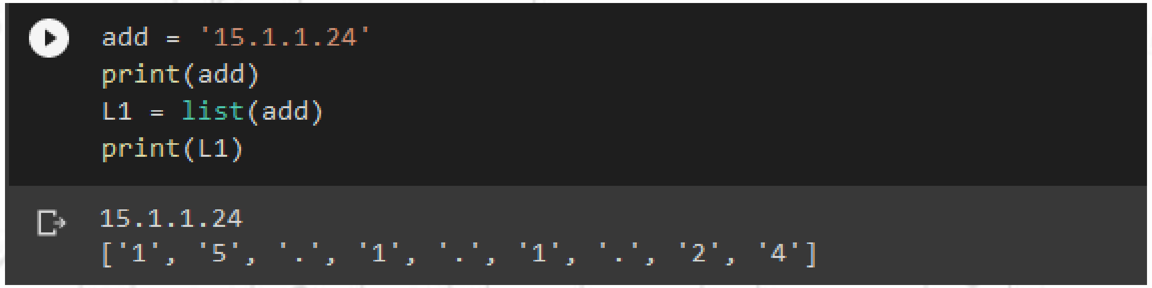
operation to change character:



```
ip = '10.1.1.2'
print(ip)
ip = ip.translate({ord('2'):'20'})
print(ip)
```

```
10.1.1.2
10.1.1.20
```

- **Converting a string into a list:**



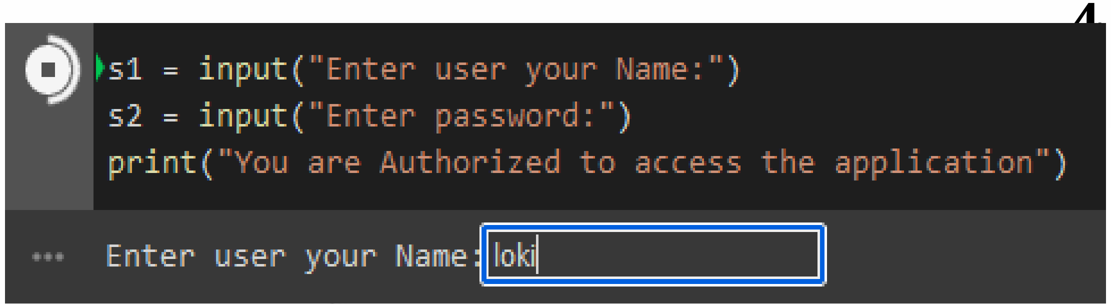
```
add = '15.1.1.24'
print(add)
L1 = list(add)
print(L1)
```

```
15.1.1.24
['1', '5', '.', '1', '.', '1', '.', '2', '4']
```

□ **Raw Input:** input **( )**:

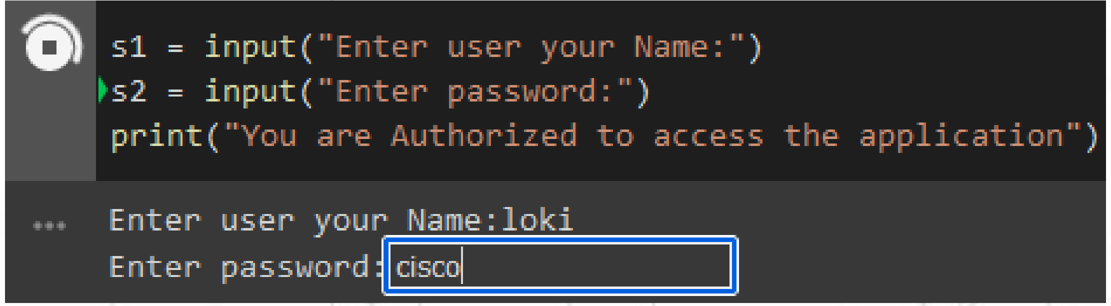
- This function first takes the input from the user and then evaluates the expression, which means Python automatically identifies whether user entered a string or a number or list. If the input provided is not correct then either syntax error or exception is raised by python. For example:

1



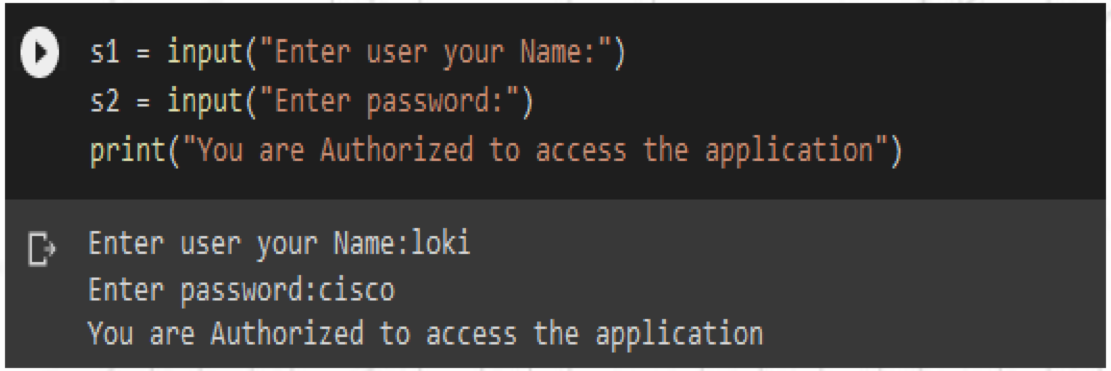
```
s1 = input("Enter user your Name:")
s2 = input("Enter password:")
print("You are Authorized to access the application")

Enter user your Name: loki
```

2



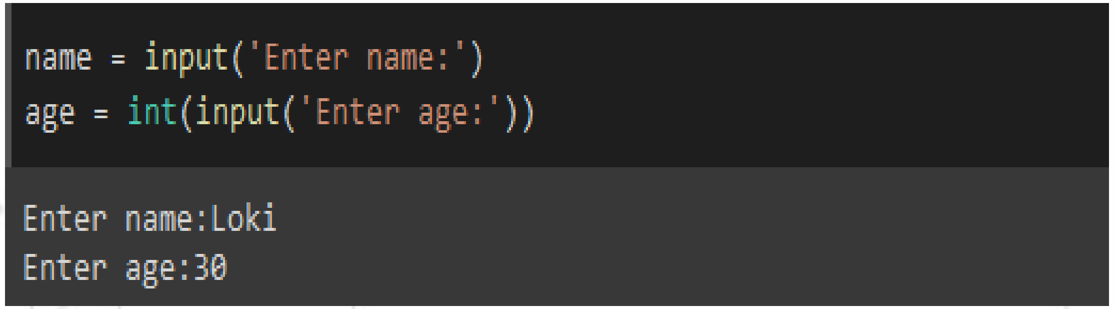
```
s1 = input("Enter user your Name:")
s2 = input("Enter password:")
print("You are Authorized to access the application")

Enter user your Name:loki
Enter password: cisco
```

3



```
s1 = input("Enter user your Name:")
s2 = input("Enter password:")
print("You are Authorized to access the application")

Enter user your Name:loki
Enter password:cisco
You are Authorized to access the application
```

4

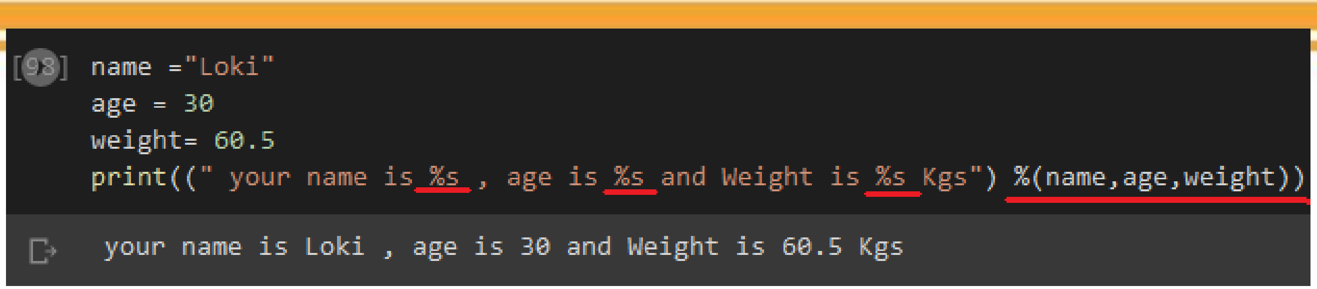


```
name = input('Enter name:')
age = int(input('Enter age:'))

Enter name:Loki
Enter age:30
```

For Integers, you need to specify as **int( )**

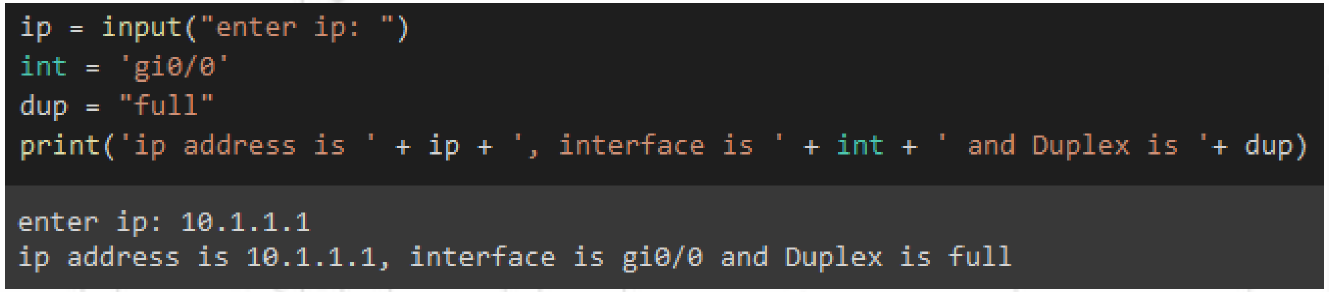
By default, the **Raw input** is treated as a string.

- **Different Methods to print variables:**



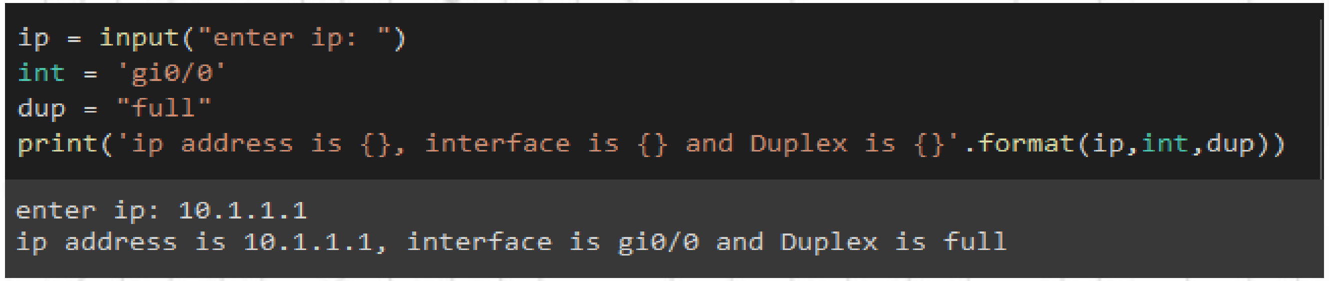
```
[93] name ="Loki"
     age = 30
     weight= 60.5
     print((" your name is %s , age is %s and Weight is %s Kgs") %(name,age,weight))

 your name is Loki , age is 30 and Weight is 60.5 Kgs
```

%s--> Placeholder so that you don’t have to break the string to insert variables n between for printing %(name,age,weight) ---> represents the variables used



```
ip = input("enter ip: ")
int = 'gi0/0'
dup = "full"
print('ip address is ' + ip + ', interface is ' + int + ' and Duplex is '+ dup)

enter ip: 10.1.1.1
ip address is 10.1.1.1, interface is gi0/0 and Duplex is full
```

Recommended method:



```
ip = input("enter ip: ")
int = 'gi0/0'
dup = "full"
print('ip address is {}, interface is {} and Duplex is {}'.format(ip,int,dup))

enter ip: 10.1.1.1
ip address is 10.1.1.1, interface is gi0/0 and Duplex is full
```

B.Lists:

- Lists are very versatile collection of Data which is ordered and changeable in Python.

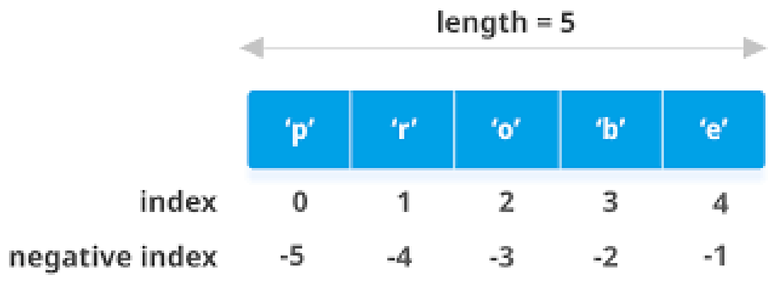
- List contains items wrapped in square brackets **“[]”** separated by commas **‘,’.**
  - Example: L1 = [A, B, C]

- Lists are similar to Arrays in other programming languages like C or Java, but there is a major difference here: There is no restriction that list should consist of same data types in Python.

- So, elements of lists can be integers, strings, other lists, tuples or any other data type as well.

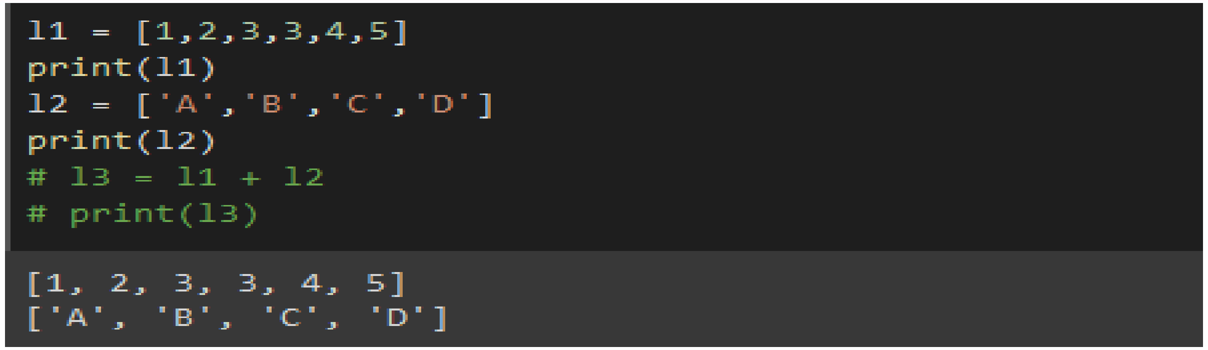
- Similar to Strings, Lists in Python are also indexed.

- List allows duplicate members. Example**: list = [A, A, B, C].**



Some of the Basic Operations you can perform on Lists.

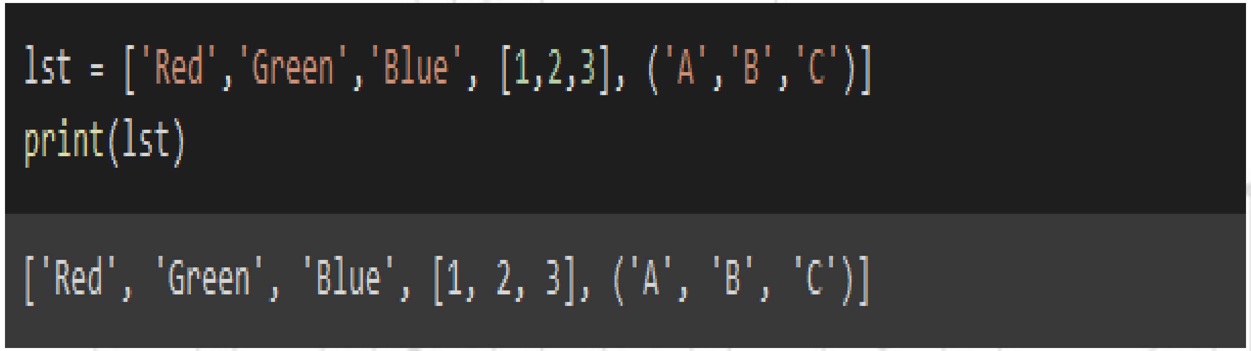
Examples & Indexing:



```
l1 = [1,2,3,3,4,5]
print(l1)
l2 = ['A','B','C','D']
print(l2)
# l3 = l1 + l2
# print(l3)
```

```
[1, 2, 3, 3, 4, 5]
['A', 'B', 'C', 'D']
```

Nested list: Lists also can contain other lists (nested lists) or tuples inside them.

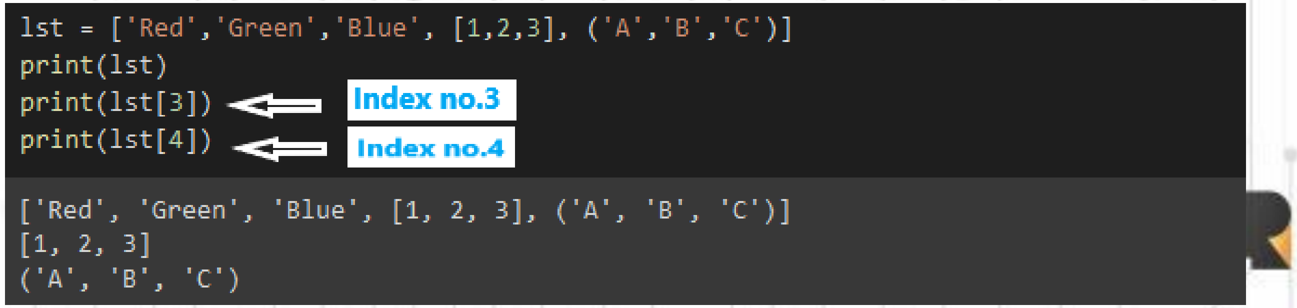


```
lst = ['Red','Green','Blue', [1,2,3], ('A','B','C')]
print(lst)
```

```
['Red', 'Green', 'Blue', [1, 2, 3], ('A', 'B', 'C')]
```

In beside the Index numbering are:

- **0 = Red**
- **1 = Green**
- **2 = Blue**
- **3 = [1,2,3]**
- **4 = (A, B, C)**



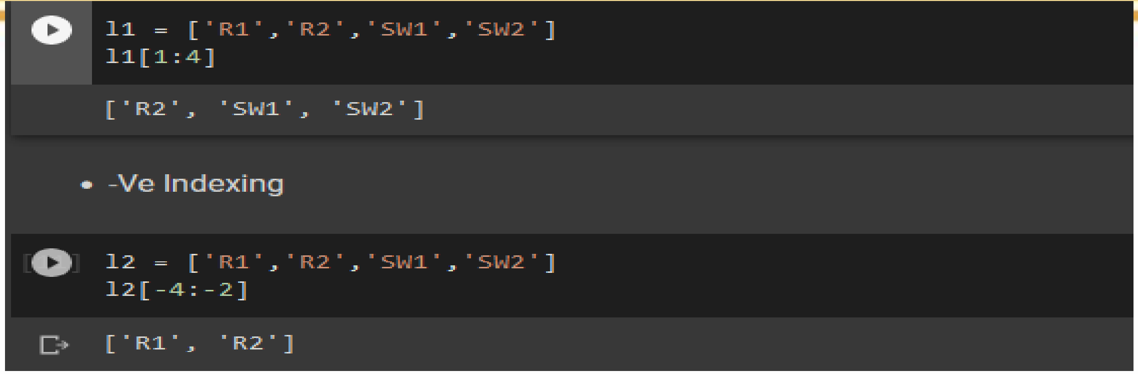
```
lst = ['Red','Green','Blue', [1,2,3], ('A','B','C')]
print(lst)
print(lst[3])
print(lst[4])
```

Index no.3

Index no.4

```
['Red', 'Green', 'Blue', [1, 2, 3], ('A', 'B', 'C')]
[1, 2, 3]
('A', 'B', 'C')
```

□ **Slicing:**



```
l1 = ['R1','R2','SW1','SW2']
l1[1:4]
```

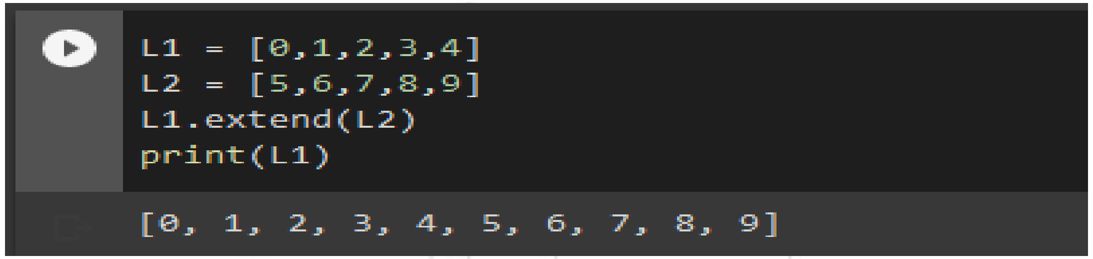
```
['R2', 'SW1', 'SW2']
```

- -Ve Indexing

```
l2 = ['R1','R2','SW1','SW2']
l2[-4:-2]
```

```
['R1', 'R2']
```

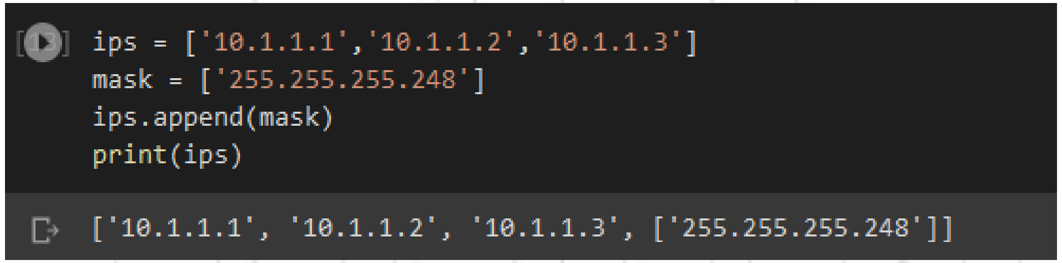
□ **Extending a List:**



```
L1 = [0,1,2,3,4]
L2 = [5,6,7,8,9]
L1.extend(L2)
print(L1)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

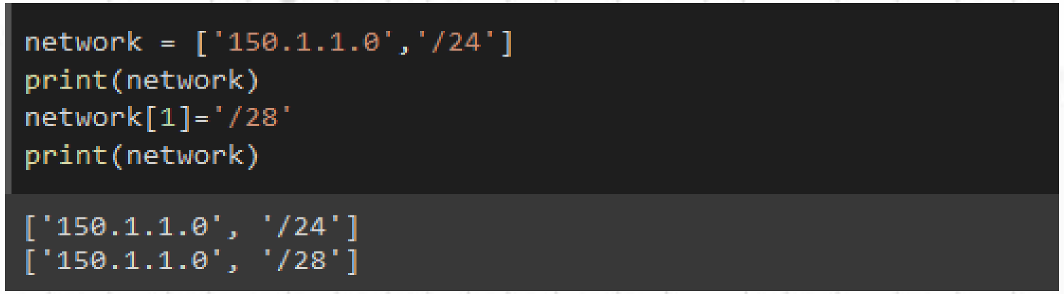
□ **Appending-> adding List to an existing on.It is not similar to extending. It adds a list at the end**



```
ips = ['10.1.1.1','10.1.1.2','10.1.1.3']
mask = ['255.255.255.248']
ips.append(mask)
print(ips)
```

```
['10.1.1.1', '10.1.1.2', '10.1.1.3', ['255.255.255.248']]
```

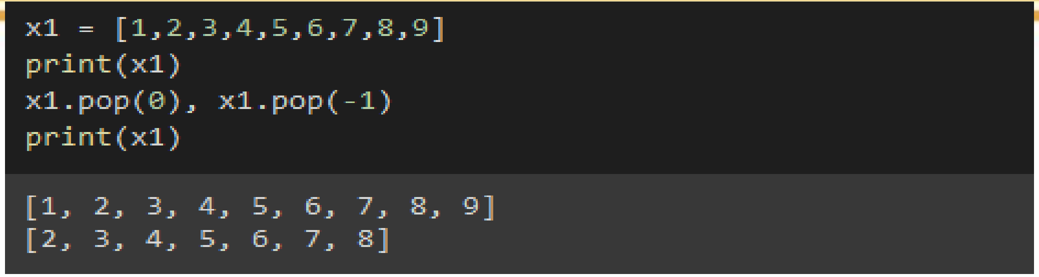
□ **Lists are Mutable: You can change a value in a List, using the Index no.**



```
network = ['150.1.1.0','/24']
print(network)
network[1]='/28'
print(network)
```

```
['150.1.1.0', '/24']
['150.1.1.0', '/28']
```

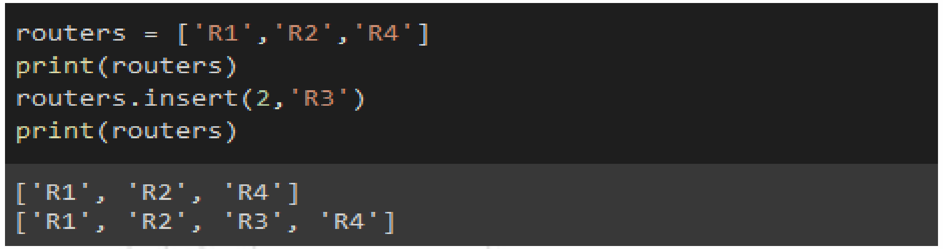
□ **Removing an Element from a List:**



```
x1 = [1,2,3,4,5,6,7,8,9]
print(x1)
x1.pop(0), x1.pop(-1)
print(x1)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[2, 3, 4, 5, 6, 7, 8]
```

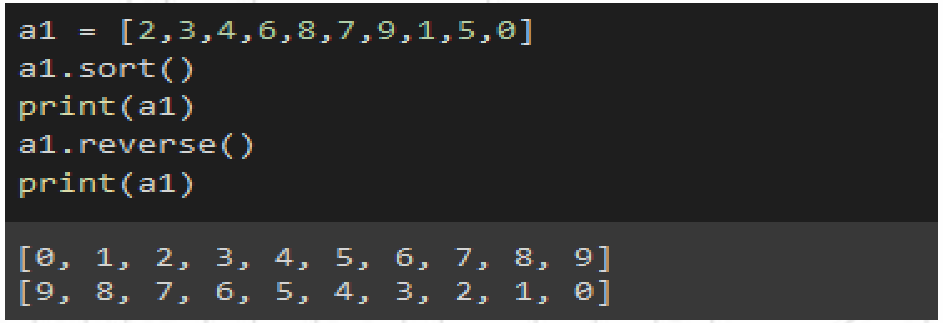
□ **Adding an Element in an existing List**



```
routers = ['R1','R2','R4']
print(routers)
routers.insert(2,'R3')
print(routers)
```

```
['R1', 'R2', 'R4']
['R1', 'R2', 'R3', 'R4']
```

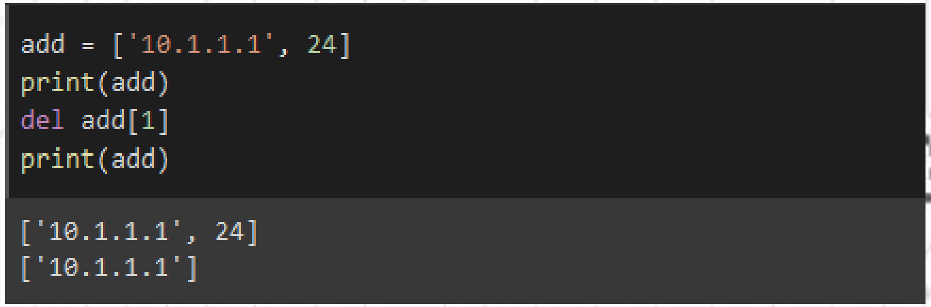
□ **Sorting and Reversing a List:**



```
a1 = [2,3,4,6,8,7,9,1,5,0]
a1.sort()
print(a1)
a1.reverse()
print(a1)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

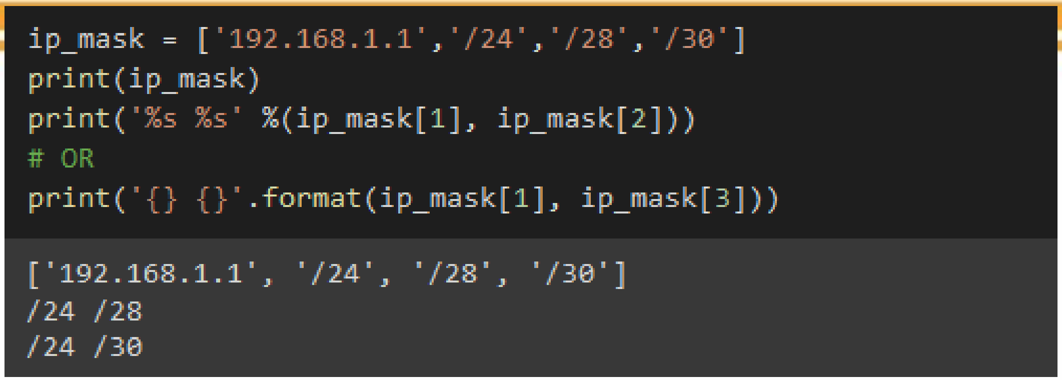
□ **Deleting an element from a list:**



```
add = ['10.1.1.1', 24]
print(add)
del add[1]
print(add)
```

```
['10.1.1.1', 24]
['10.1.1.1']
```

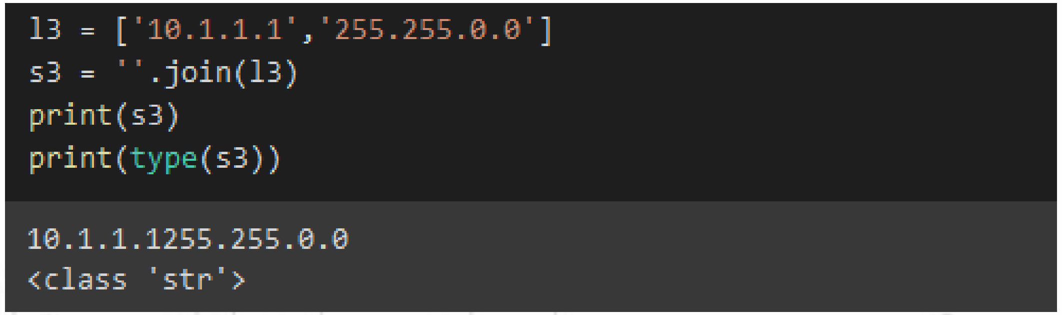
□ Printing only selected Elements in a List:



```
ip_mask = ['192.168.1.1','/24','/28','/30']
print(ip_mask)
print('%s %s' %(ip_mask[1], ip_mask[2]))
# OR
print('{} {}'.format(ip_mask[1], ip_mask[3]))
```

```
['192.168.1.1', '/24', '/28', '/30']
/24 /28
/24 /30
```

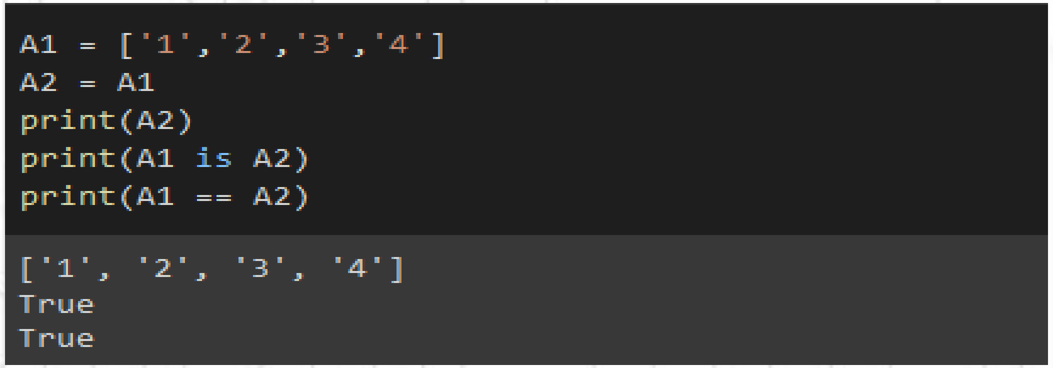
□ Converting a List into a string:



```
l3 = ['10.1.1.1','255.255.0.0']
s3 = ''.join(l3)
print(s3)
print(type(s3))
```

```
10.1.1.1255.255.0.0
<class 'str'>
```

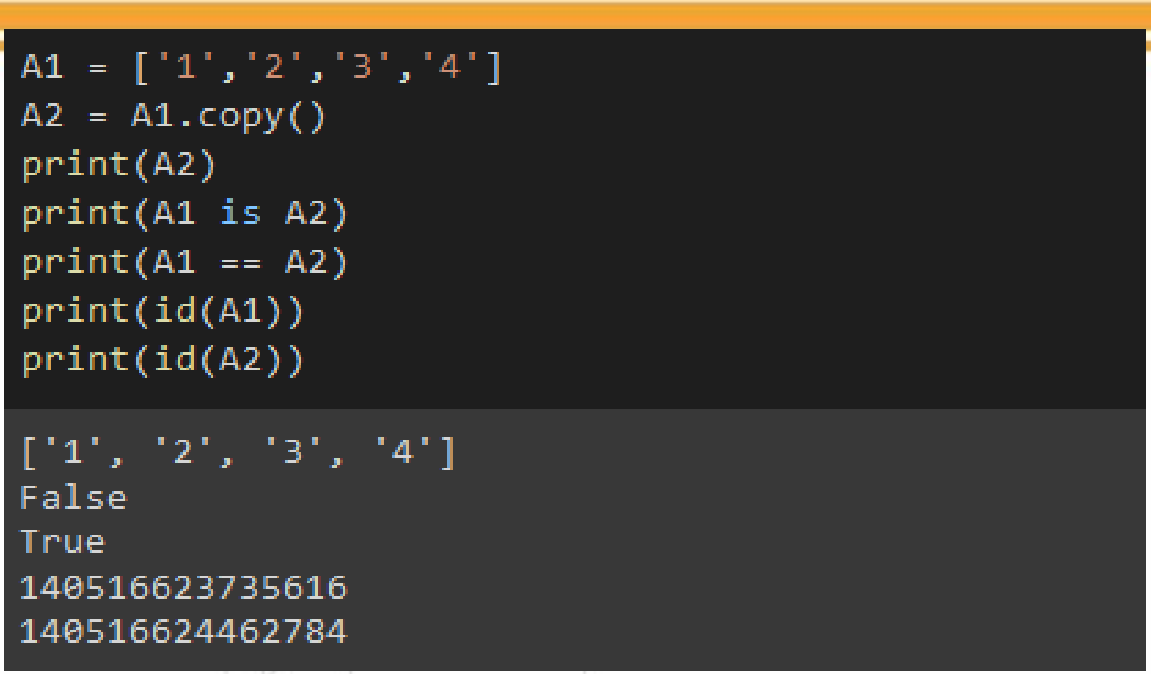
□ Copy a List by reference:



```
A1 = ['1','2','3','4']
A2 = A1
print(A2)
print(A1 is A2)
print(A1 == A2)
```

```
['1', '2', '3', '4']
True
True
```

□ Copy a List by Operations:

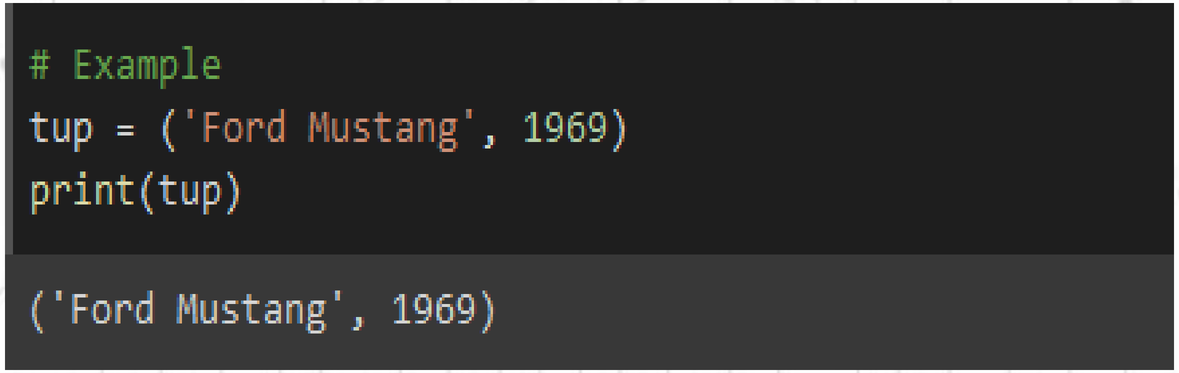


```
A1 = ['1','2','3','4']
A2 = A1.copy()
print(A2)
print(A1 is A2)
print(A1 == A2)
print(id(A1))
print(id(A2))
```

```
['1', '2', '3', '4']
False
True
140516623735616
140516624462784
```

Tuple

- A tuple is a collection of objects(data) which are ordered and immutable.
- Tuples are sequences, just like lists and are Immutable.
- The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.
- A tuple consists of a number of values separated by commas and enclosed in **( )**

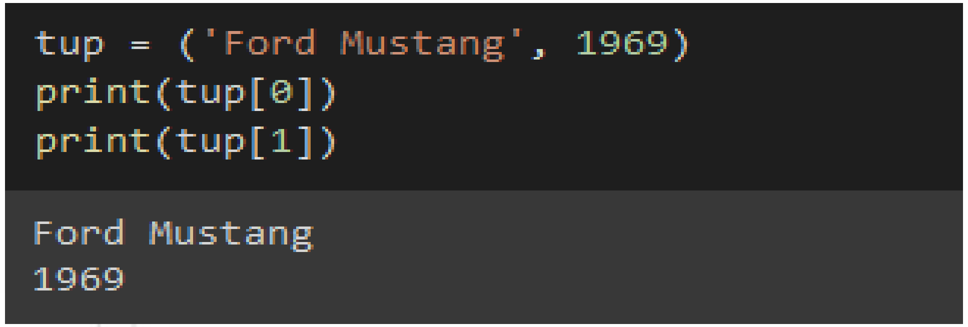


```
# Example
tup = ('Ford Mustang', 1969)
print(tup)
```

```
('Ford Mustang', 1969)
```

Some of the Basic Operations you can perform on Tuple.

□ **Indexing: Tuples are also Indexed.**

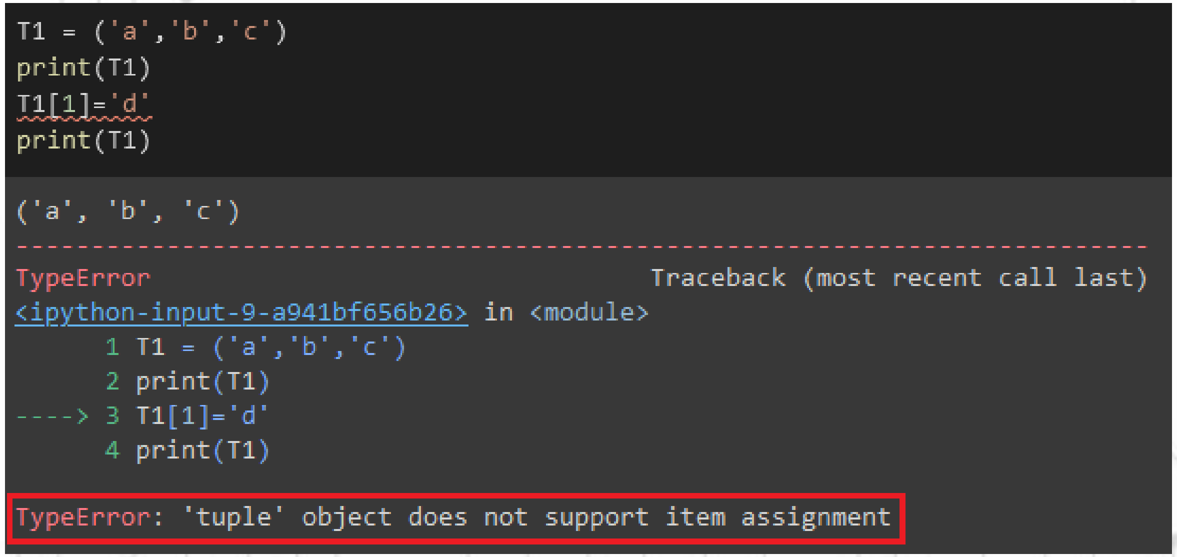


```
tup = ('Ford Mustang', 1969)
print(tup[0])
print(tup[1])
```

```
Ford Mustang
1969
```

□ **Tuples are Immutable:**

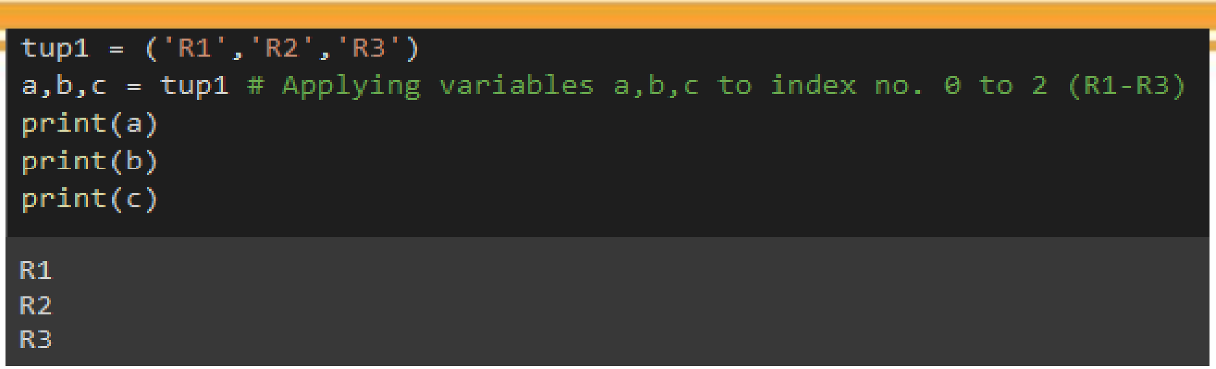
You can't change elements once it is assigned in a Tuple.



```
T1 = ('a','b','c')
print(T1)
T1[1]='d'
print(T1)
```

```
('a', 'b', 'c')
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-9-a941bf656b26> in <module>
      1 T1 = ('a','b','c')
      2 print(T1)
----> 3 T1[1]='d'
      4 print(T1)

TypeError: 'tuple' object does not support item assignment
```

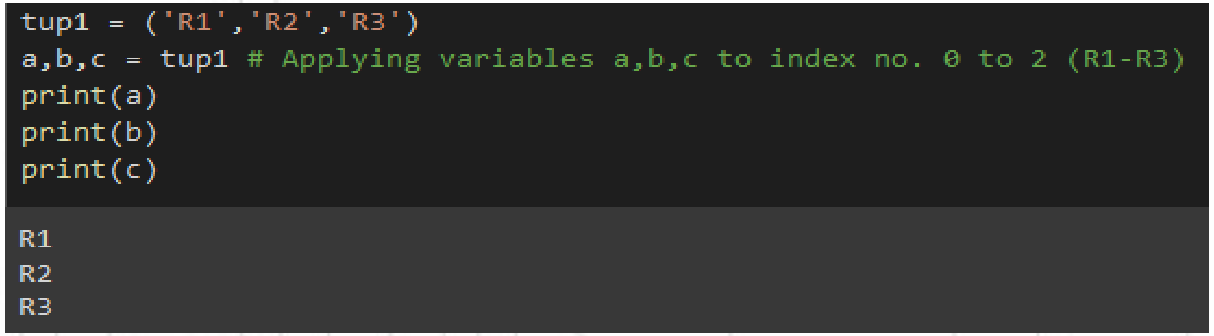
- □ **You can apply multiple variables to a Tuple: Tuple Unpacking**



```
tup1 = ('R1','R2','R3')
a,b,c = tup1 # Applying variables a,b,c to index no. 0 to 2 (R1-R3)
print(a)
print(b)
print(c)
```

```
R1
R2
R3
```

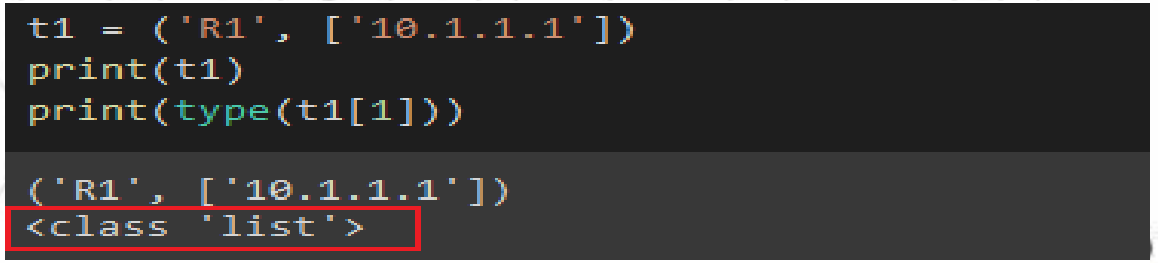
- □ **You can apply multiple variables to a Tuple: Tuple Unpacking**



```
tup1 = ('R1','R2','R3')
a,b,c = tup1 # Applying variables a,b,c to index no. 0 to 2 (R1-R3)
print(a)
print(b)
print(c)
```

```
R1
R2
R3
```

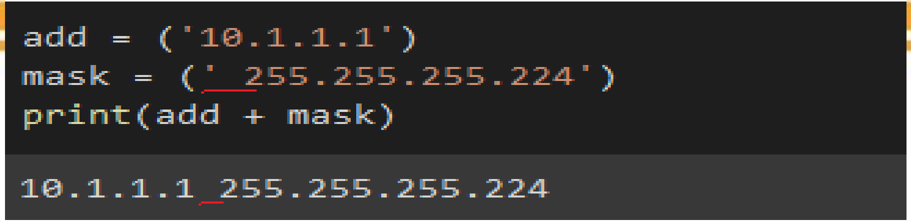
- □ **Nested Tuple: With List**



```
t1 = ('R1', ['10.1.1.1'])
print(t1)
print(type(t1[1]))
```

```
('R1', ['10.1.1.1'])
<class 'list'>
```

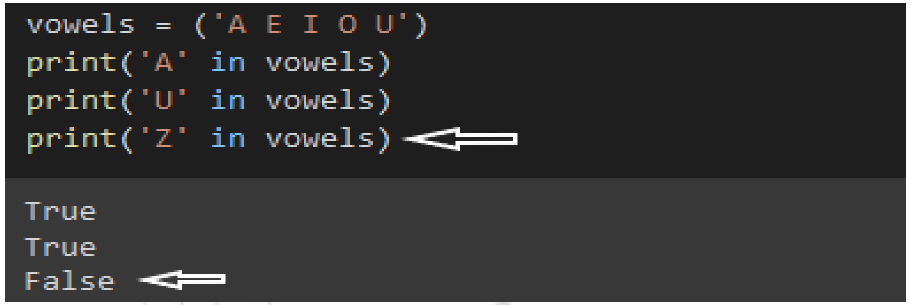
- □ **Concatenation in Tuple**



```
add = ('10.1.1.1')
mask = (' 255.255.255.224')
print(add + mask)
```

```
10.1.1.1 255.255.255.224
```

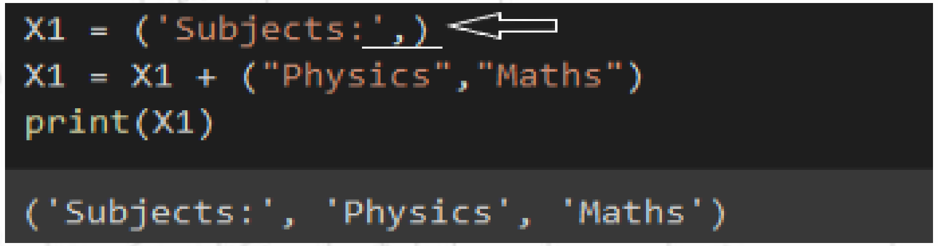
- □ **Tuple Membership Test:**



```
vowels = ('A E I O U')
print('A' in vowels)
print('U' in vowels)
print('Z' in vowels)
```

```
True
True
False
```

- □ **Creating a tuple with a single element:**

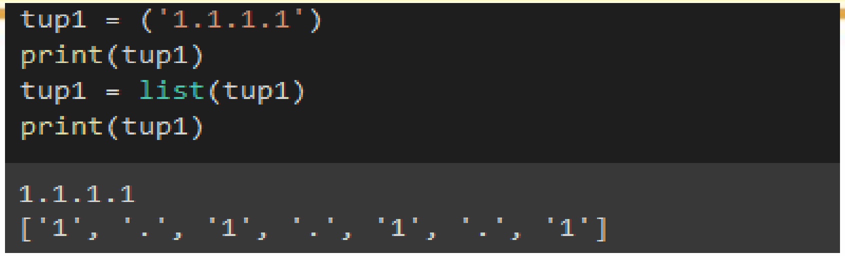


```
X1 = ('Subjects:',)
X1 = X1 + ("Physics","Maths")
print(X1)
```

```
('Subjects:', 'Physics', 'Maths')
```

The comma (,) after the single element is a must, which implies that this tuple can be continued.

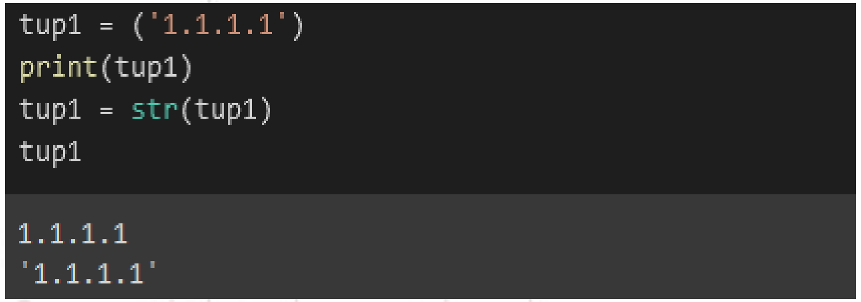
□ **Converting a Tuple into a List:**



```
tup1 = ('1.1.1.1')
print(tup1)
tup1 = list(tup1)
print(tup1)
```

```
1.1.1.1
['1', '.', '1', '.', '1', '.', '1']
```

□ **Converting a Tuple into a String:**



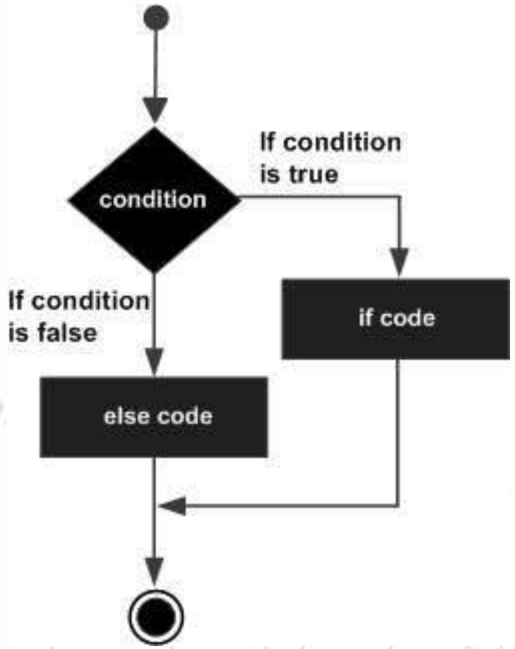
```
tup1 = ('1.1.1.1')
print(tup1)
tup1 = str(tup1)
tup1
```

```
1.1.1.1
'1.1.1.1'
```

Dictionaries

Decision Making

- Normal execution flow of the program statements is top to bottom
- But there could be situations where we need to alter this sequential flow of execution
- One such situation is Decision Making
- We would need to decide which code block to execute depending on satisfying certain conditions
- Makes use of a data type known as Boolean which consists of only two values - True and False

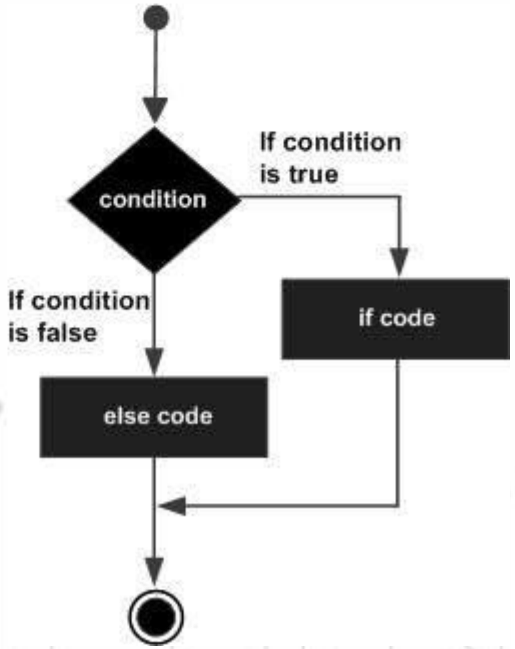


Conditionals

- In Python decision making is achieved using conditional statements
- Used to tell the program what to do when the condition is evaluated to True

If statement

```
if expression:
 statement
```



Ansible

For Network Automation

Introduction

- Ansible is a configuration management tool
- Ansible scripts are known as playbooks
- Playbooks consists of plays which are nothing but collection of tasks to be performed as part of configuration management
- Ansible consists of a control node and a bunch of managed nodes

Introduction (Cont.)

- Control node is where Ansible is installed and managed nodes are the systems on which we want to perform some tasks
- Ansible uses push-based model i.e. tasks are pushed by the control node onto the target nodes
- Ansible is agentless i.e. we do not need to install any agent on the target nodes

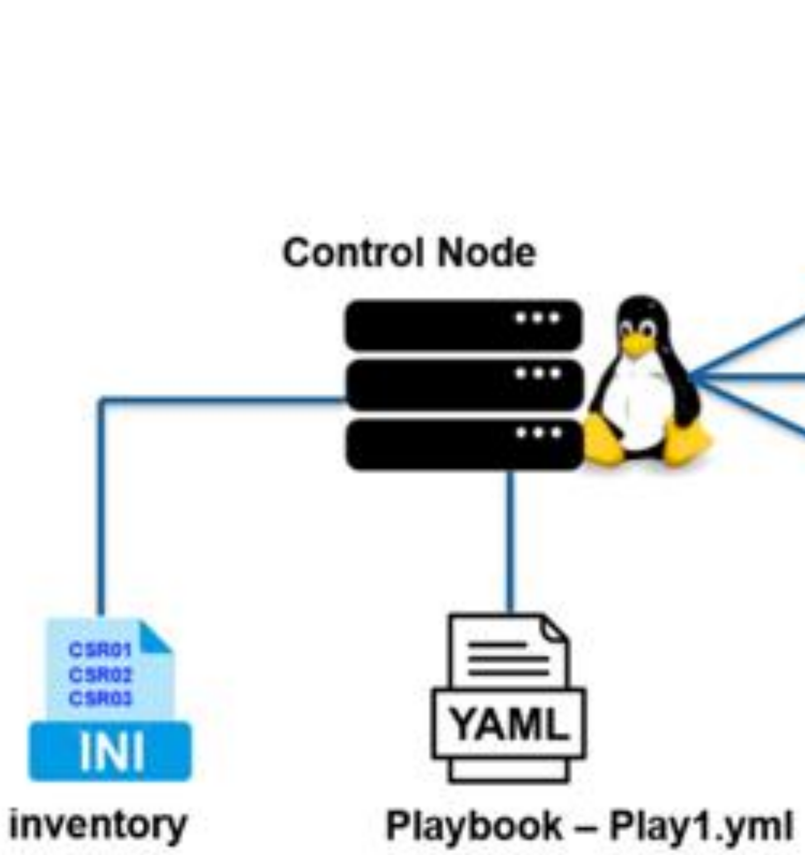
Ansible Components

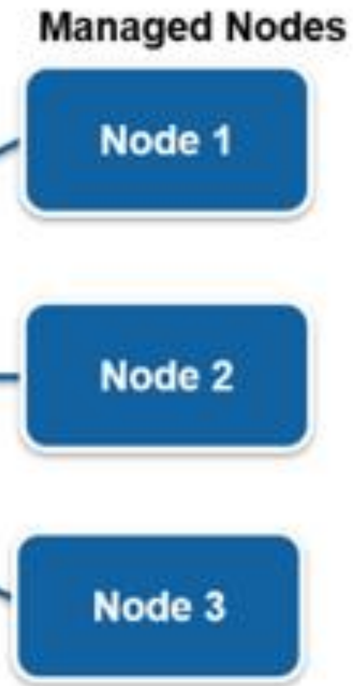
- Control Node
  - Linux machine with Python and Ansible installed which is used to manage remote Linux servers or other devices
  - We cannot use the windows machine as a control node
  - Use multiple control nodes for resiliency

Ansible Components (Cont.)

- Managed Nodes
  - These nodes are the devices which are managed by Ansible control node
  - This can be Linux servers or Networking devices
  - We do not need Ansible to be installed on Managed Nodes

Ansible Components (Cont.)





IP ROUTE TECHNOLOGIES PVT LTD

Automating Linux Servers

- Uses SSH to connect to the server
- Server does not have Ansible installed
- Copies Python code to the server (server must have Python execution engine)
- Server executes code and returns status of tasks

Automating Network Devices

- Python code runs locally on the Ansible control host (where Ansible is installed)
- Equivalent of writing Python scripts on a single server
- No code is copied to the device.
- Device does not need to have Python

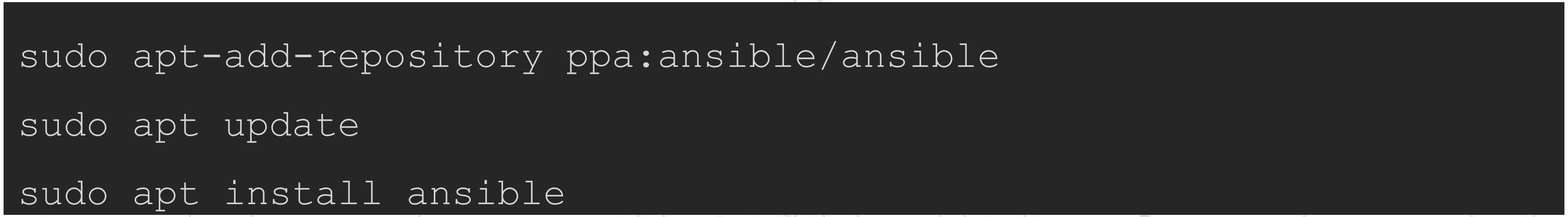
Ansible Installation

- Using Linux package managers like apt or rpm etc.
  - Recommended for beginners
- Using Python package manager pip
  - Recommended for advanced users. Gives more control on the Python version and Python packages used by Ansible
- Follow the following link for detailed instructions to install Ansible
  - https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-ansible-on-ubuntu-20-04

IP ROUTE
TECHNOLOGIES PVT LTD

Ansible Installation (Cont.)

- Installation steps are shown below



```
sudo apt-add-repository ppa:ansible/ansible

sudo apt update

sudo apt install ansible
```

Ansible Installation (Cont.)

- If installation is successful, we can check the version using the following command

```
$ ansible --version
ansible [core 2.12.10]
  config file = /home/nexadmin/work/ccna_devnet/ansible/ansible.cfg
  configured module search path = ['/home/nexadmin/.ansible/plugins/modules',
'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location =
/home/nexadmin/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.8.10 (default, Nov 14 2022, 12:59:47) [GCC 9.4.0]
  jinja version = 2.10.1
  libyaml = True
```

Ansible Configuration

- Ansible needs a bunch of settings to work like SSH settings, location where modules are located, etc.
- Default values have been specified for all the settings that Ansible needs
- However, they can be customised by specifying them in a config file called ansible.cfg

Ansible Configuration (Cont.)

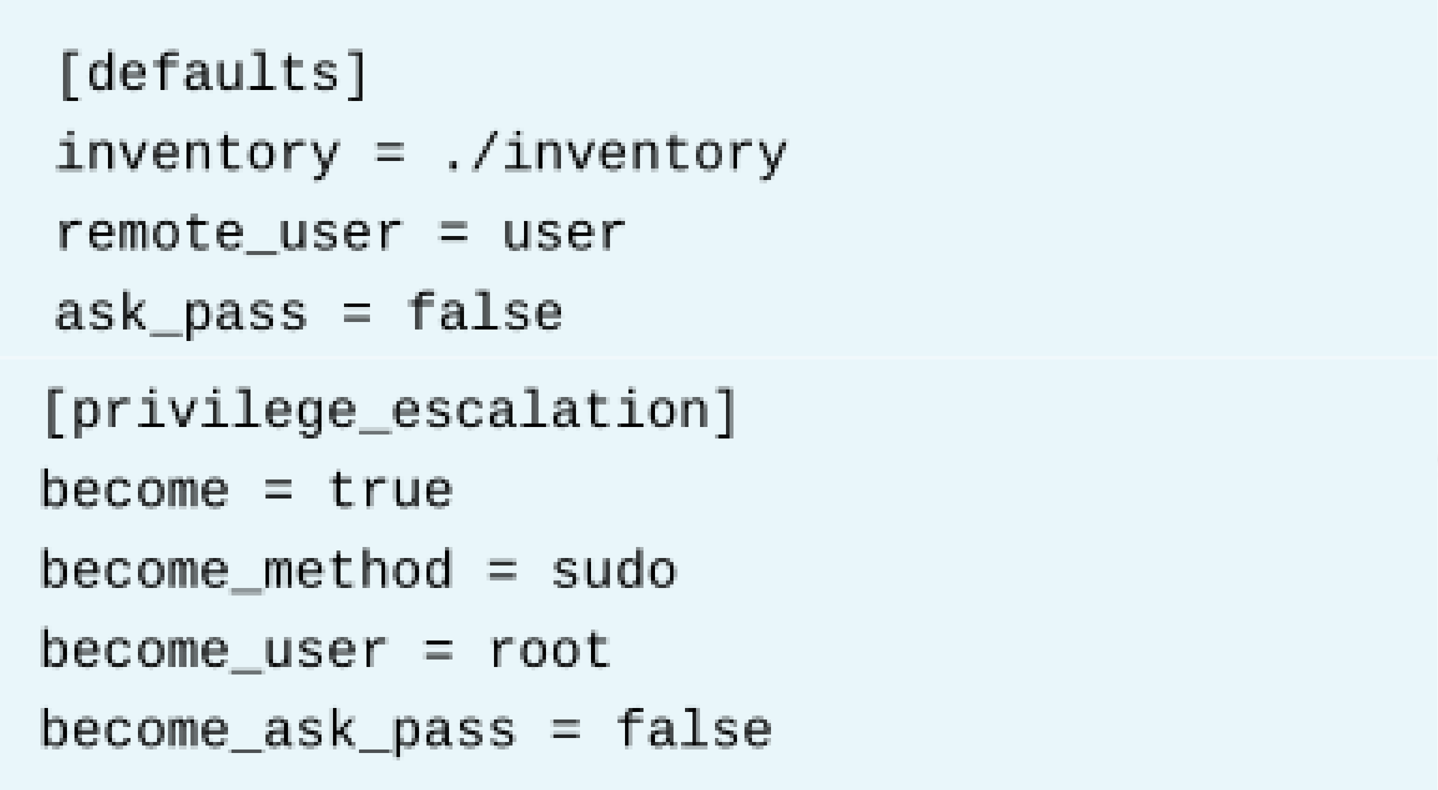
- By default, Ansible looks for ansible.cfg in the following locations
  - ANSIBLE_CONFIG env variable, if set
  - Current directory, where Ansible commands are being executed
  - Logging in user’s home directory
  - In /etc/ansible directory (default location with a default ansible.cfg created by the installer)

Ansible Configuration (Cont.)

- Ansible recommends keeping ansible.cfg in project root dir so that settings can be customised for each project
- The output of the command ansible --version shows which ansible.cfg is being used

Ansible Configuration (Cont.)

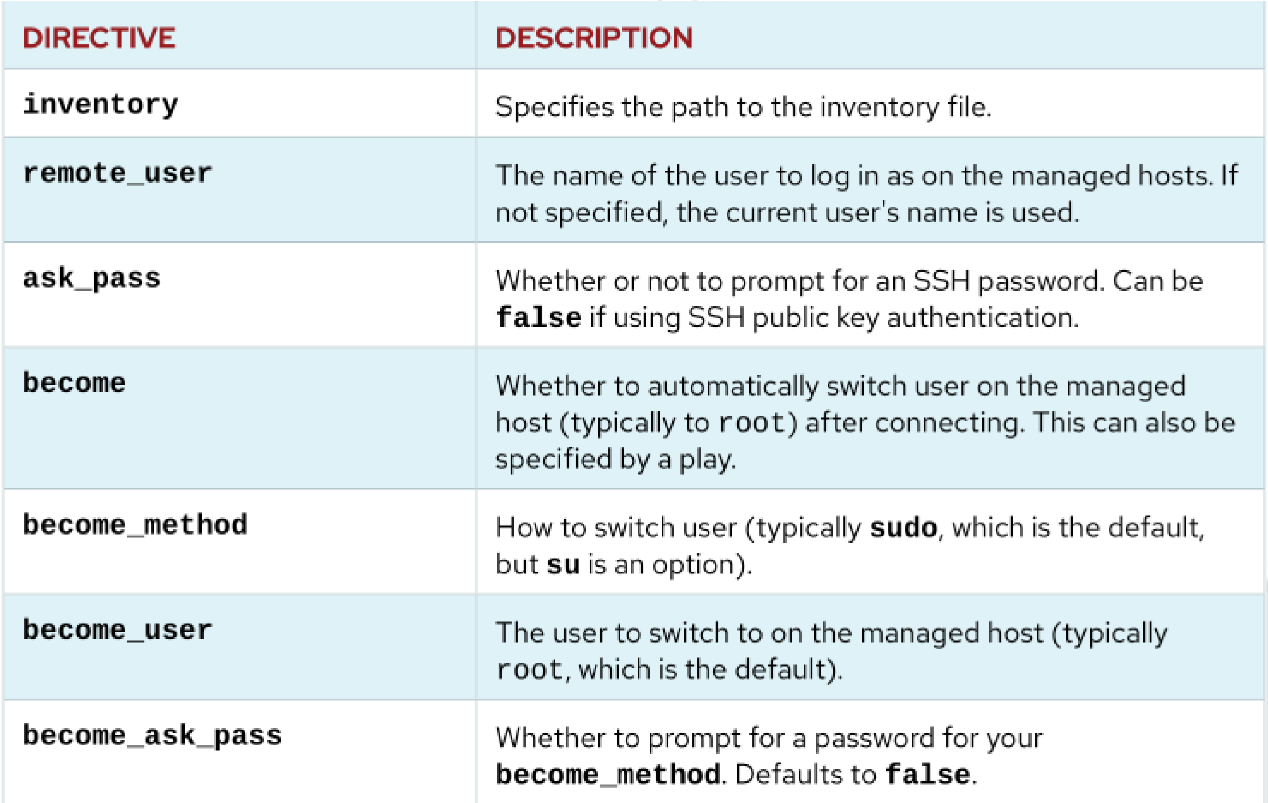
- Shown below is a typical config file along with description of what each entry stands for



```
[defaults]
inventory = ./inventory
remote_user = user
ask_pass = false
[privilege_escalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = false
```

Ansible Configuration (Cont.)

- Description of commonly used config settings is as follows



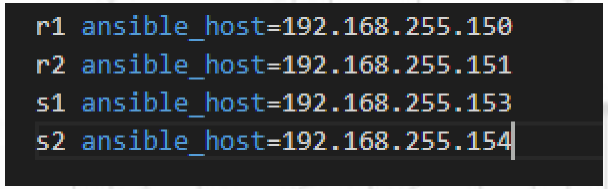
DIRECTIVE	DESCRIPTION
`inventory`	Specifies the path to the inventory file.
`remote_user`	The name of the user to log in as on the managed hosts. If not specified, the current user's name is used.
`ask_pass`	Whether or not to prompt for an SSH password. Can be **`false`** if using SSH public key authentication.
`become`	Whether to automatically switch user on the managed host (typically to `root`) after connecting. This can also be specified by a play.
`become_method`	How to switch user (typically **`sudo`**, which is the default, but **`su`** is an option).
`become_user`	The user to switch to on the managed host (typically `root`, which is the default).
`become_ask_pass`	Whether to prompt for a password for your **`become_method`**. Defaults to **`false`**.

Ansible Inventory

- Inventory in Ansible is the list of devices being managed by the Control node
- The file which contains this list of devices is known as inventory file and is another important file
- Default file name and location of Ansible inventory is ***/etc/ansible/hosts***
- This can also be changed by creating inventory files per project or as needed
- If we change the name and location of inventory file, the same needs to be mentioned in ansible.cfg so that Ansible knows where to look for inventory

Ansible Inventory (Cont.)

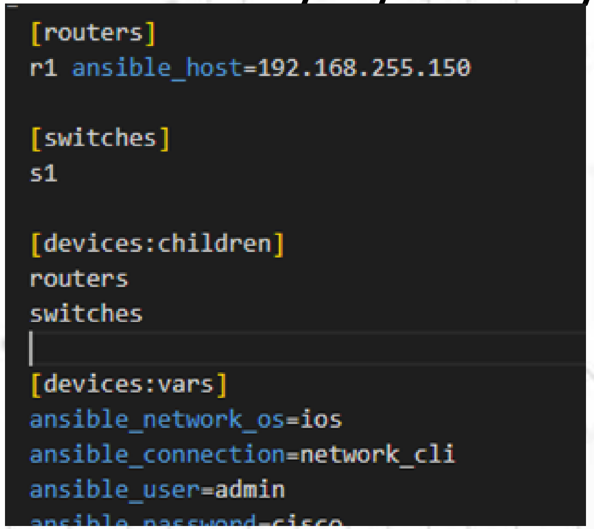
- Inventory can be specified either in INI file format or YAML file format
- Shown below is a typical inventory consisting of few routers and few switches



```
r1 ansible_host=192.168.255.150
r2 ansible_host=192.168.255.151
s1 ansible_host=192.168.255.153
s2 ansible_host=192.168.255.154
```

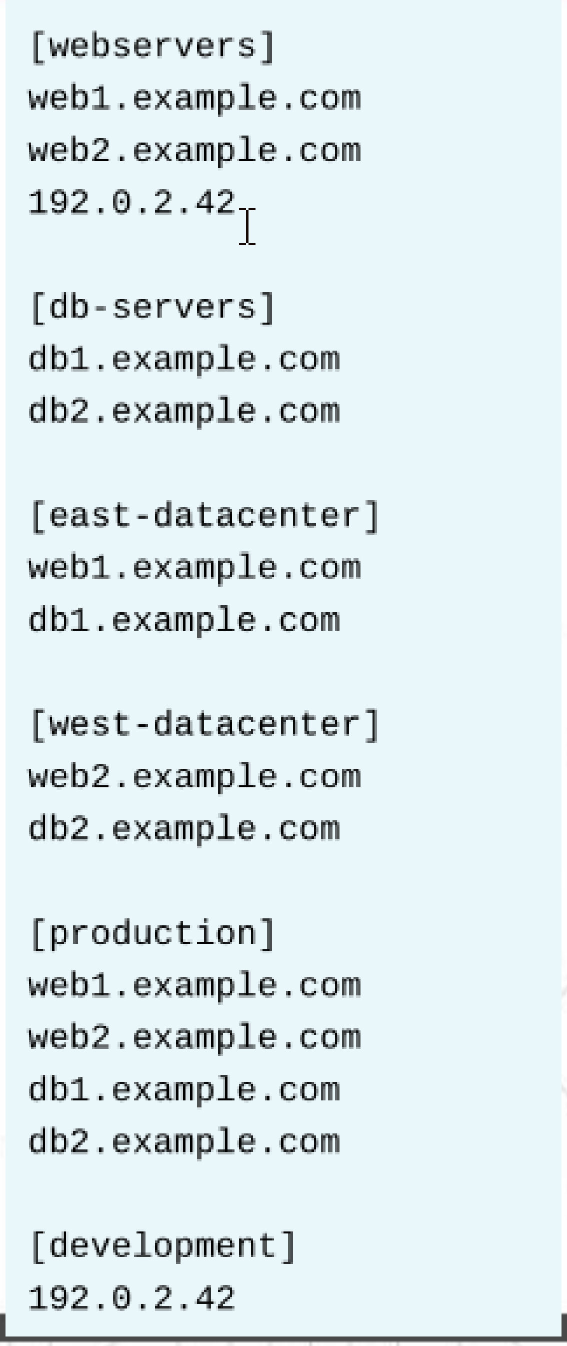
Ansible Inventory (Cont.)

- While this works fine, the kind of config steps for all routers, switches would be similar
- Config settings could also differ region wise, division wise etc.
- So we can group hosts in inventory by their type, division, region or any other logical separation



```
[routers]
r1 ansible_host=192.168.255.150

[switches]
s1

[devices:children]
routers
switches

[devices:vars]
ansible_network_os=ios
ansible_connection=network_cli
ansible_user=admin
ansible_password=cisco
```

Grouping example



```
[webservers]
web1.example.com
web2.example.com
192.0.2.42

[db-servers]
db1.example.com
db2.example.com

[east-datacenter]
web1.example.com
db1.example.com

[west-datacenter]
web2.example.com
db2.example.com

[production]
web1.example.com
web2.example.com
db1.example.com
db2.example.com

[development]
192.0.2.42
```

Getting Started

- Create a new file in the current dir or home dir called ansible.cfg
- Put the following contents in it

```
[defaults]

inventory=/home/project1/hosts
```

Getting Started (Cont.)

- Create a new file called ***inventory*** and put the following contents in it

```
R1 ansible_host=192.168.255.150 ansible_network_os=ios
S1 ansible_host=192.168.255.153 ansible_network_os=ios
```

Getting Started (Cont.)

- Use the following command to check if the inventory is correctly identified by Ansible

```
$ ansible-inventory --list
```

```
{
 "_meta": {
          "hostvars": {
          "r1": {
                    "ansible_host": "192.168.255.150",
                    "ansible_network_os": "ios"
          },
          "s1": {
                    "ansible_host": "192.168.255.153",
                    "ansible_network_os": "ios"
          }
          }
          }
```

Getting Started (Cont.)

- Ansible also uses variables to store data similar to programming languages
- In the above example, ***ansible_host***, ***ansible_network_os*** are the variables
- These are defined at each individual host level, hence they are called as **Host Vars**
- Hosts in inventory can be divided into related groups by type like routers, switches or by geography apac, emea
- Common attributes can be set at group level known as **Group Vars**

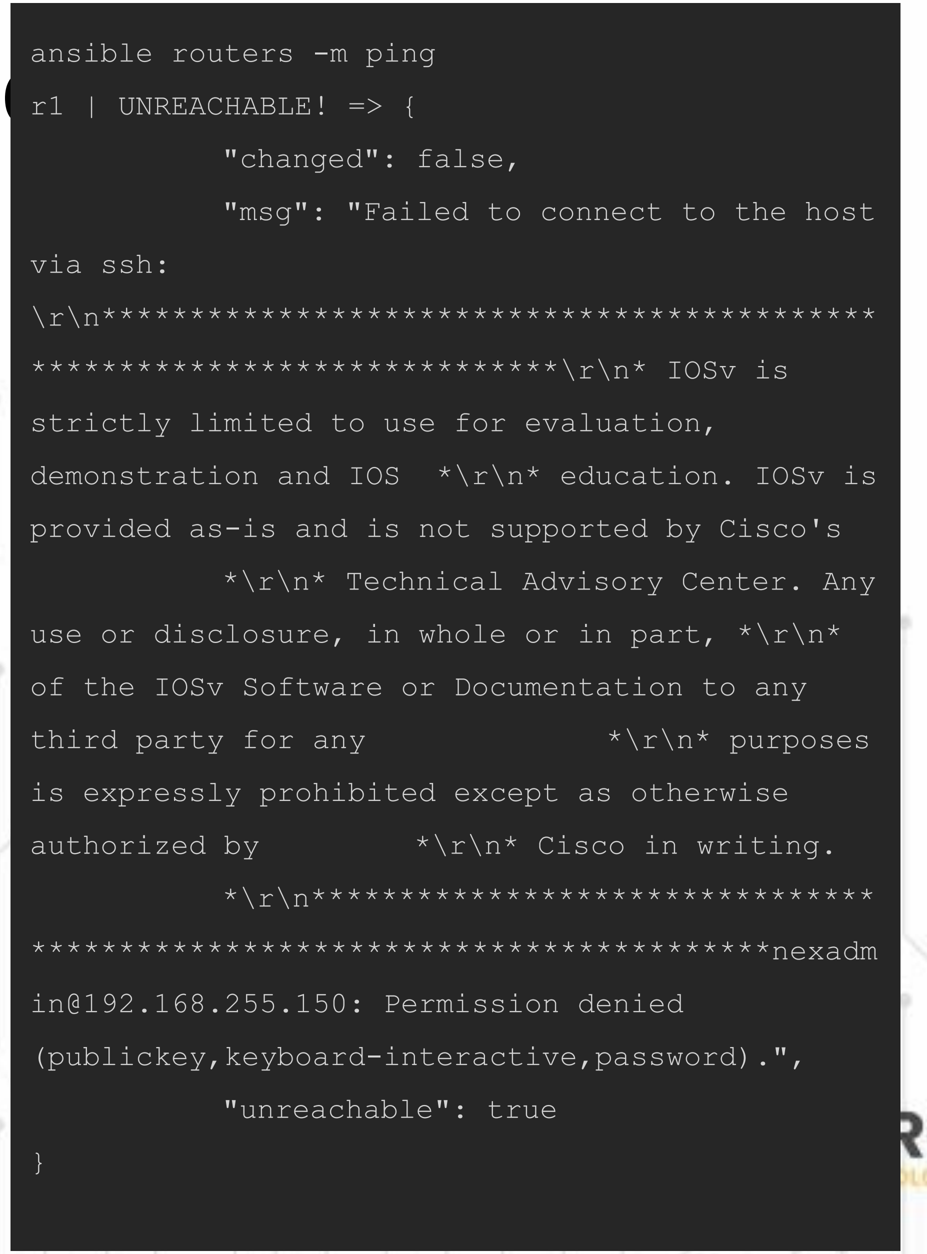
Getting Started (Cont.)

- We can rewrite inventory file as follows

```
[routers]
r1 ansible_host=192.168.255.150

[switches]
s1 ansible_host=192.168.255.153

[devices:children]
routers
switches

[devices:vars]
ansible_network_os=ios
```

Getting Start

- Let us check if the devices are reachable using ping. This is not icmp ping, but ansible module ping.
- We should get an output as follows



```
ansible routers -m ping
r1 | UNREACHABLE! => {
           "changed": false,
           "msg": "Failed to connect to the host
via ssh:
\r\n********************************************
******************************\r\n* IOSv is
strictly limited to use for evaluation,
demonstration and IOS  *\r\n* education. IOSv is
provided as-is and is not supported by Cisco's
           *\r\n* Technical Advisory Center. Any
use or disclosure, in whole or in part, *\r\n*
of the IOSv Software or Documentation to any
third party for any              *\r\n* purposes
is expressly prohibited except as otherwise
authorized by         *\r\n* Cisco in writing.
           *\r\n********************************
******************************************nexadm
in@192.168.255.150: Permission denied
(publickey,keyboard-interactive,password).",
           "unreachable": true
}
```

Getting Started (Cont.)

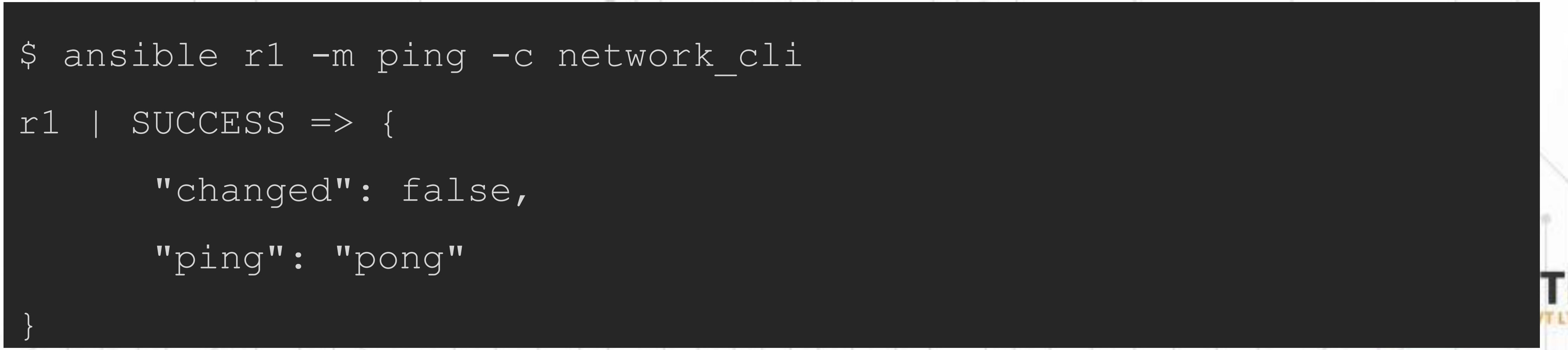
- Ansible default connect mode is ssh
- In the above command we just asked Ansible to ping the devices
- So it tries to use ssh to connect to the devices and then perform ping
- But we have not specified any other details like ssh user or password
- We have not added ssh keys also to be able to connect without requiring username and password

Getting Started (Cont.)

- Even after providing username and password it fails because, as we noted earlier, ping is not an icmp ping but a simple python command which connects to the device and returns a message “pong”
- More details here
  - https://serverfault.com/questions/1107102/ansible-ping-fail-session-request-sent-but-read-header-failed-broken-pi

Getting Started (Cont.)

- So workaround is to use connection mode as network_cli
- Even better way is to use net_ping which we will use while writing playbooks



```
$ ansible r1 -m ping -c network_cli
r1 | SUCCESS => {
      "changed": false,
      "ping": "pong"
}
```

Ansible Ad hoc Commands

- The command we have used above to ping the managed nodes is known as an ad hoc command
- Ad hoc commands are command given directly on the terminal without putting them in scripts (playbooks)
- For performing simple non-repetitive tasks ad hoc commands are quite handy
- However, for doing anything significant in Ansible, it is preferred to follow the playbook approach as it gives more flexibility and reusability

Ansible Ad hoc Commands (Cont.)

- Check version of the network device using “show version” command

```
$ ansible r1 -m cli_command -a “command=’show version’”
```

- Above command uses a module called ***cli_command*** which can be used to issue ad hoc commands with cli based devices

Ansible Ad hoc Commands (Cont.)

```
r1 | SUCCESS => {
      "ansible_facts": {
      "discovered_interpreter_python": "/usr/bin/python3"
      },
      "changed": false,
      "stdout": "Cisco IOS Software, IOSv Software (VIOS-
ADVENTERPRISEK9-M), Version 15.6(2)T, RELEASE SOFTWARE
(fc2)\nTechnical Support:
http://www.cisco.com/techsupport\nCopyright (c) 1986-2016 by Cisco
Systems, Inc.
```

Ansible Ad hoc Commands (Cont.)

- Check interface details of the network device using “sh ip int br” command

```
$ ansible r1 -m ios_command -a “commands=’sh ip int br’”
```

- Above command uses a module called ***ios_command*** which is module in Ansible for using ad hoc commands with specific type of devices, in this cse ios devices

Ansible Ad hoc Commands (Cont.)

```
"stdout_lines": [
      [
            "Interface                    IP-Address         OK?
Method Status                 Protocol",
            "GigabitEthernet0/0           192.168.255.150 YES NVRAM
up                     up    ",
            "GigabitEthernet0/1           unassigned         YES
NVRAM  administratively down down   ",
            "GigabitEthernet0/2           unassigned         YES
NVRAM  administratively down down   ",
```

Ansible Ad hoc Commands (Cont.)

- Create an empty file on the linux managed host

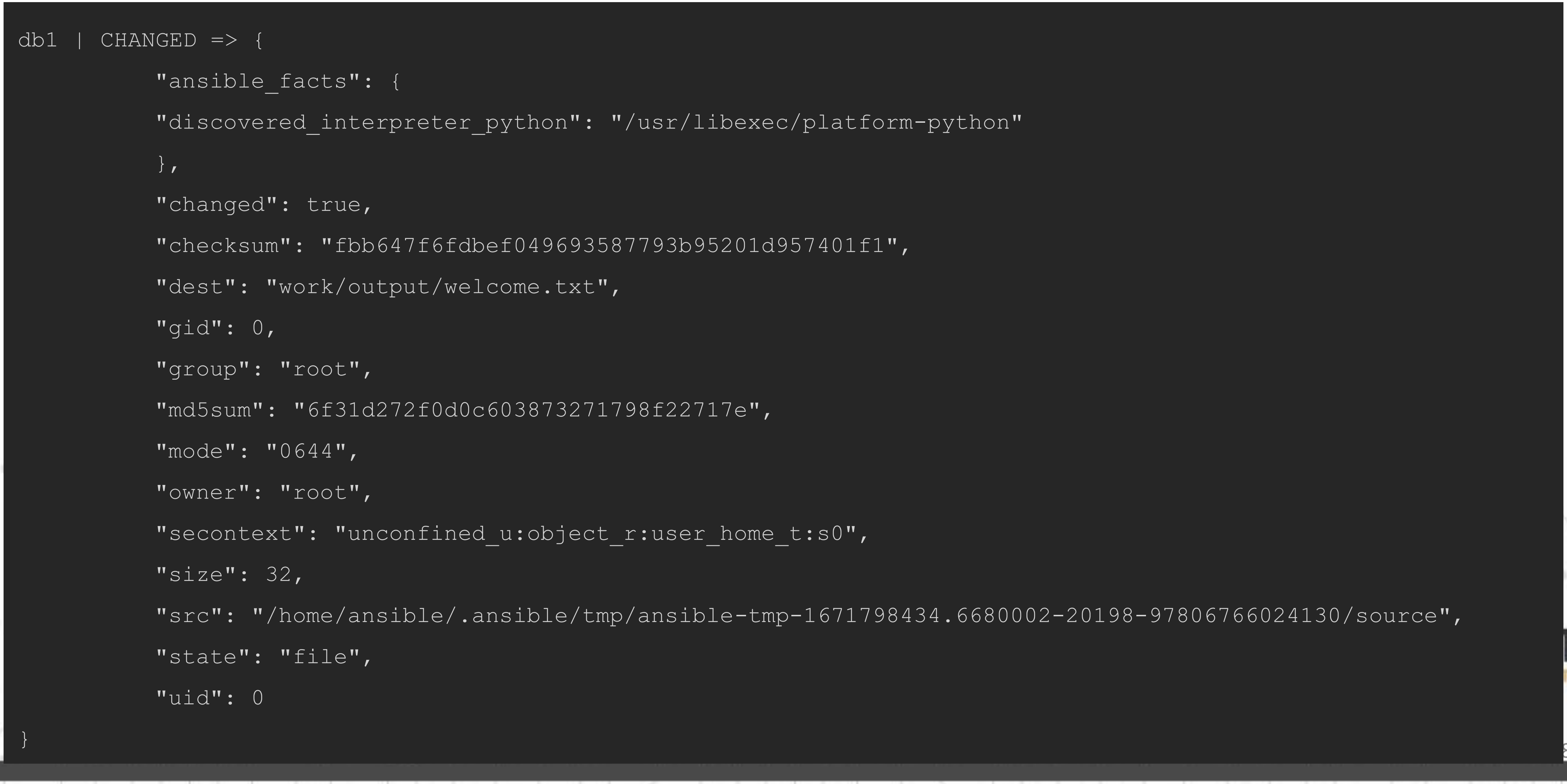
```
$ ansible db1 -m command -a "touch work/output/welcome.txt"

db1 | CHANGED | rc=0 >>
```

Ansible Ad hoc Commands (Cont.)

- Edit the file we just created and put some content in it using the copy module

```
$ ansible db1 -m copy -a "content='Welcome to CCNA DevNet
Training\n' dest='work/output/welcome.txt'"
```

Ansible Ad hoc Commands (Cont.)



```
db1 | CHANGED => {
          "ansible_facts": {
          "discovered_interpreter_python": "/usr/libexec/platform-python"
          },
          "changed": true,
          "checksum": "fbb647f6fdbef049693587793b95201d957401f1",
          "dest": "work/output/welcome.txt",
          "gid": 0,
          "group": "root",
          "md5sum": "6f31d272f0d0c603873271798f22717e",
          "mode": "0644",
          "owner": "root",
          "secontext": "unconfined_u:object_r:user_home_t:s0",
          "size": 32,
          "src": "/home/ansible/.ansible/tmp/ansible-tmp-1671798434.6680002-20198-97806766024130/source",
          "state": "file",
          "uid": 0
}
```

Ansible Playbooks

- A ***play*** is an ordered set of ***tasks*** run against ***hosts*** selected from your ***inventory***
- A playbook is a text file containing a list of one or more plays to run in a specific order
- Lengthy manual administrative steps can be broken down into structured plays which can be used repeatedly against managed hosts
- Plays can also act as documented state of your IT infrastructure
- Playbook is normally saved with .yml (or .yaml) extension

Playbook Indentation

- Data elements at the same level in the hierarchy (such as items in the same list) must have the same indentation
- Items that are children of another item must be indented more than their parents
- You can also add blank lines for readability which get ignored when file is executed
- Only the space character can be used for indentation; tab characters are not allowed

Writing a Playbook

- A playbook begins with a line consisting of three dashes (---) as a start of document marker
- It may end with three dots (...). This optional and often omitted
- The plays and tasks are executed in the same order as they are mentioned in the playbook

Playbook Example

```
$ ansible r1 -m cli_command -a "command='sh ip int br'"
```

```
---
- name: playbook to get interface details using cli_command
  hosts: r1

  tasks:
    - name: using cli_command to run show interface command
      cli_command:
        command: sh ip int br
      register: command_output

    - name: print command output
      debug:
        var: command_output
```

Running a Playbook

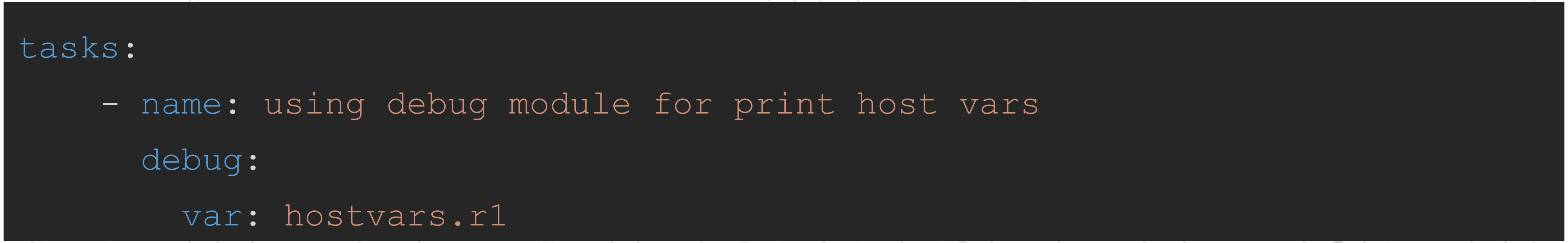
- We use the command ansible-playbook to run the playbooks
- When you run the playbook, output is generated to show the play and tasks being executed
- Output also reports the results of each task executed

Running Playbook

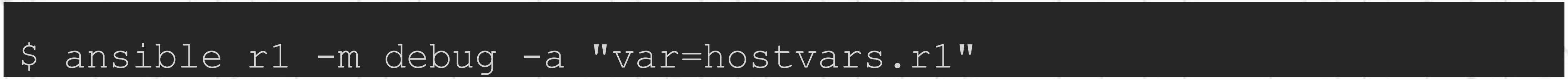
```
PLAY [playbook to get interface details using cli_command] ************************
TASK [using cli_command to run show interface command]
***********************************************************************************
ok: [r1]

TASK [print command output]
***********************************************************************************
ok: [r1] => {
       "command_output": {
       "changed": false,
       "failed": false,
       "stdout": "Interface  IP-Address     OK? Method Status
       Protocol\nGigabitEthernet0/0         192.168.255.150 …………….
"stdout_lines": [
               "Interface                    IP-Address     OK? Method Status
       Protocol",
               "GigabitEthernet0/0 192.168.255.150 YES NVRAM  up           up
       ",
               "GigabitEthernet0/1 unassigned       YES NVRAM  administratively
down down
               "GigabitEthernet0/2 unassigned       YES NVRAM  administratively
down down
```

Special Variables - hostvars

- The hostvars is a special variable which is associated with each host in the inventory and contains the list of variables associated with that host



```
tasks:
    - name: using debug module for print host vars
      debug:
        var: hostvars.r1
```



```
$ ansible r1 -m debug -a "var=hostvars.r1"
```

Special Variables - ansible_version

- When we tried to print hostcars associated with a specific host, we saw a bunch of other data that got displayed
- Some of it is useful and hence can be extracted as needed
- One such is ansible_version and can be displayed as below

```
tasks:
    - name: using debug module for print host vars
      debug:
        var: ansible_version
```

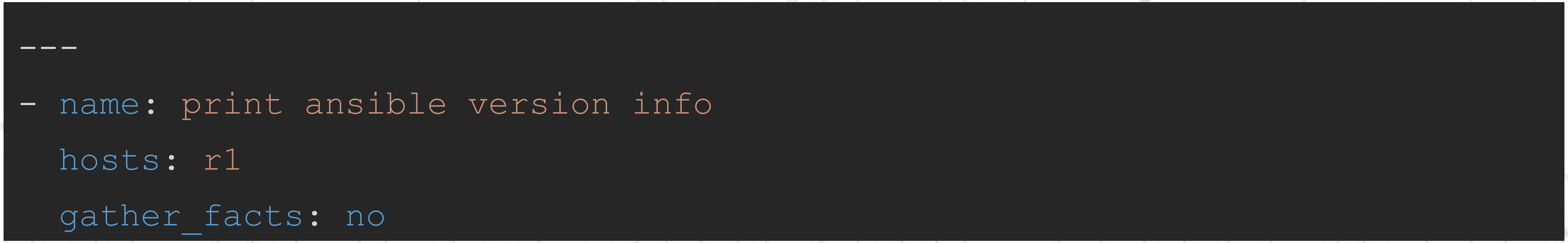
Special Variables - ansible_facts

- When we are running playbooks, there is a little task getting called without being explicitly called - known as Gathering Facts
- Ansible facts are again a bunch of useful data stored in the form of variables and available for each host

```
tasks:
    - name: using debug module for print ansible facts
      debug:
        var: ansible_facts
```

Special Variables - ansible_facts (Cont.)

- Most of the times, we would not be doing anything with all the data being fetched by Ansible in the form of ansible_facts
- Hence it can be disabled to save playbook execution time



```
---
- name: print ansible version info
  hosts: r1
  gather_facts: no
```

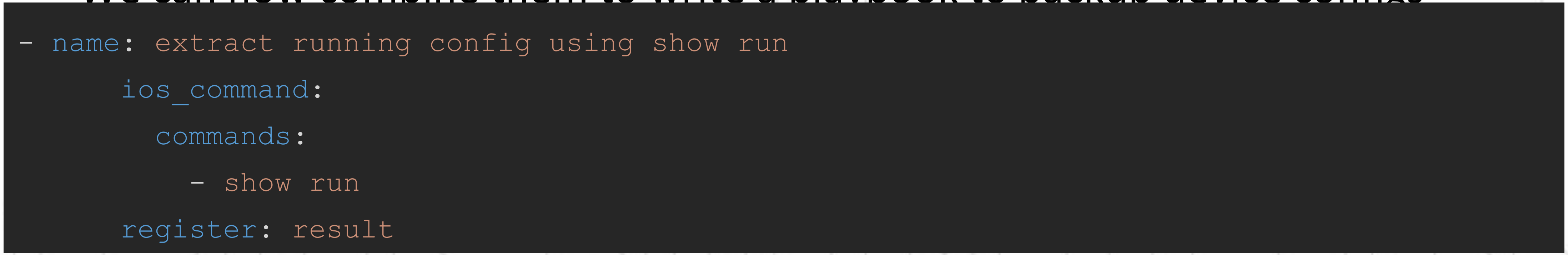
Debug Module to Print Output

- We can use debug module to also print any message to the console
- For this instead of the ***var*** parameter, we can use ***msg*** parameter

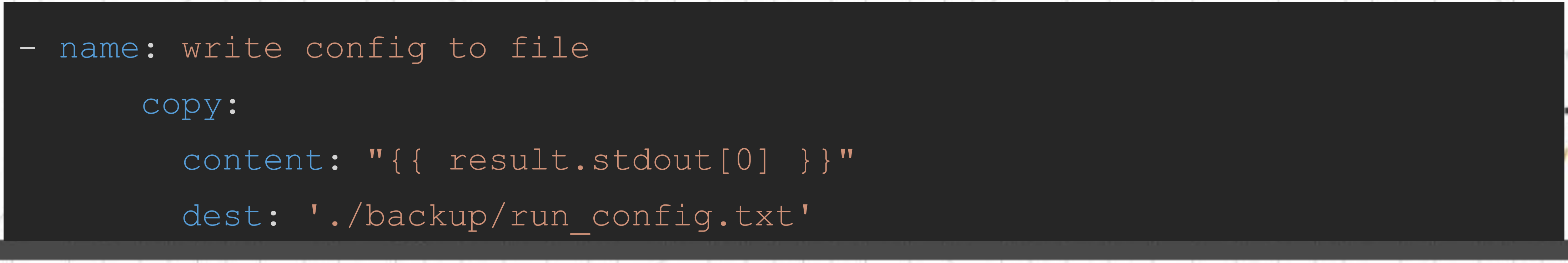
```
tasks:
    - name: using debug module for print host vars
      debug:
        msg: '''Hello... Welcome to CCNA DevNet training...
              The ansible version we are using is ---
{{hostvars.r1.ansible_version.full}}'''
```

Backing Up Configs

- We saw how to run commands on devices using commands module and copy content to files using copy module
- We can now combine them to write a playbook to backup device configs



```
- name: extract running config using show run
      ios_command:
        commands:
          - show run
      register: result
```



```
- name: write config to file
      copy:
        content: "{{ result.stdout[0] }}"
        dest: './backup/run_config.txt'
```

Ansible Modules

- We can use andble-doc command or online links to check all the modules available in Ansible
- https://docs.ansible.com/ansible/2.9/modules/list_of_network_modules.html

```
$ ansible-doc -l
```

IOS Configuration

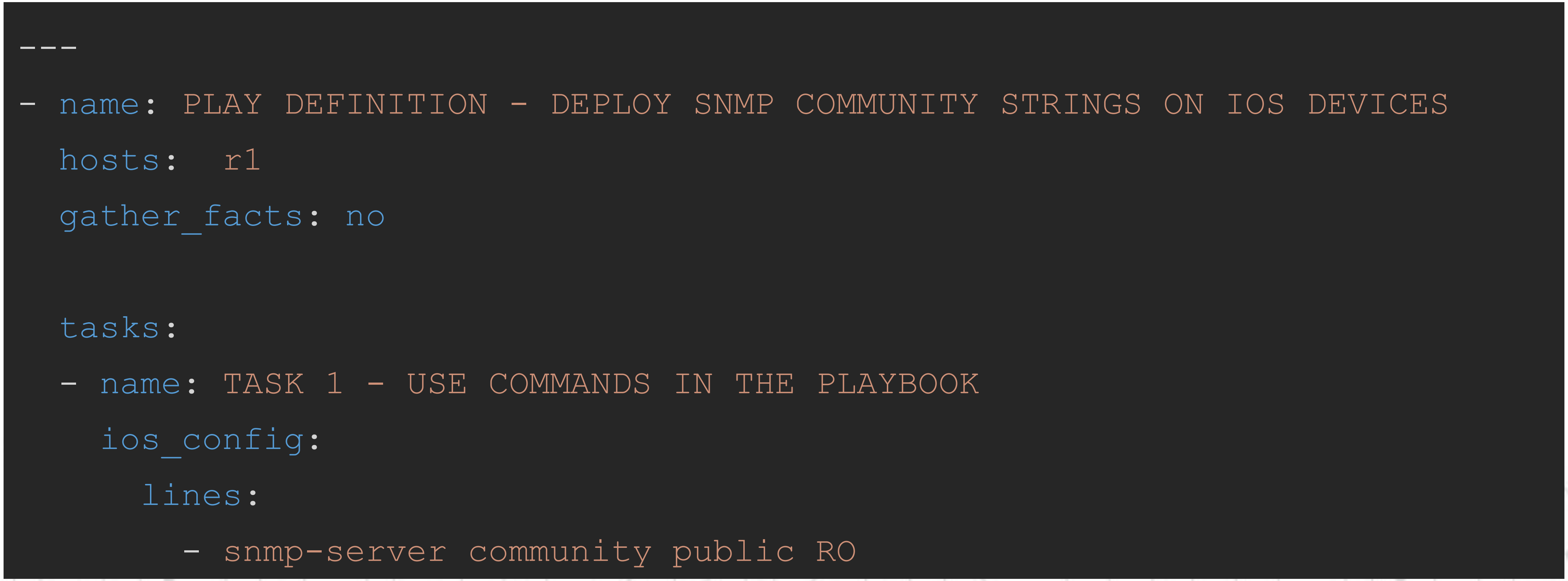
- So far we have used ios_command module to run various run commands on Cisco IOS devices
- Let us see how we can run config commands using ios_config module

```
$ ansible-doc ios_config
> CISCO.IOS.IOS_CONFIG  (/usr/lib/python3/dist-
packages/ansible_collections/cisco/ios/plugins/modules/ios_config.py)

        Cisco IOS configurations use a simple block indent file syntax for segmenting
configuration into
        sections.  This module provides an implementation for working with IOS
configuration sections in a
        deterministic way
```

IOS Configuration (Cont.)

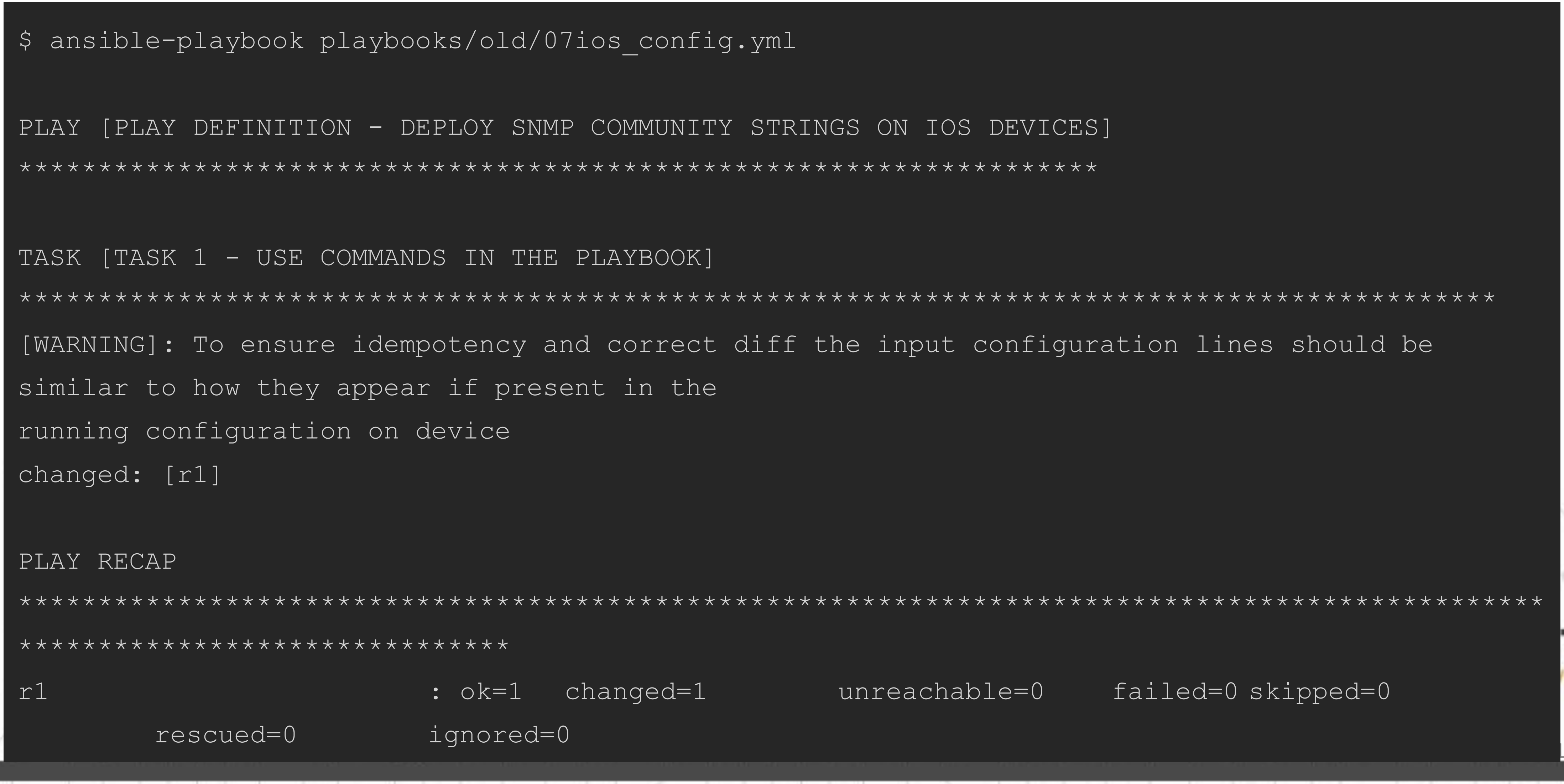
- Deploying SNMP Community Strings on Cisco router
  - The “SNMP community string” is like a user ID or password that allows access to a router’s stats
- We will use ios_config module for this task

IOS Configuration (Cont.)



```
---
- name: PLAY DEFINITION - DEPLOY SNMP COMMUNITY STRINGS ON IOS DEVICES
  hosts:  r1
  gather_facts: no

  tasks:
  - name: TASK 1 - USE COMMANDS IN THE PLAYBOOK
    ios_config:
      lines:
        - snmp-server community public RO
```

IOS Configuration (Cont.)

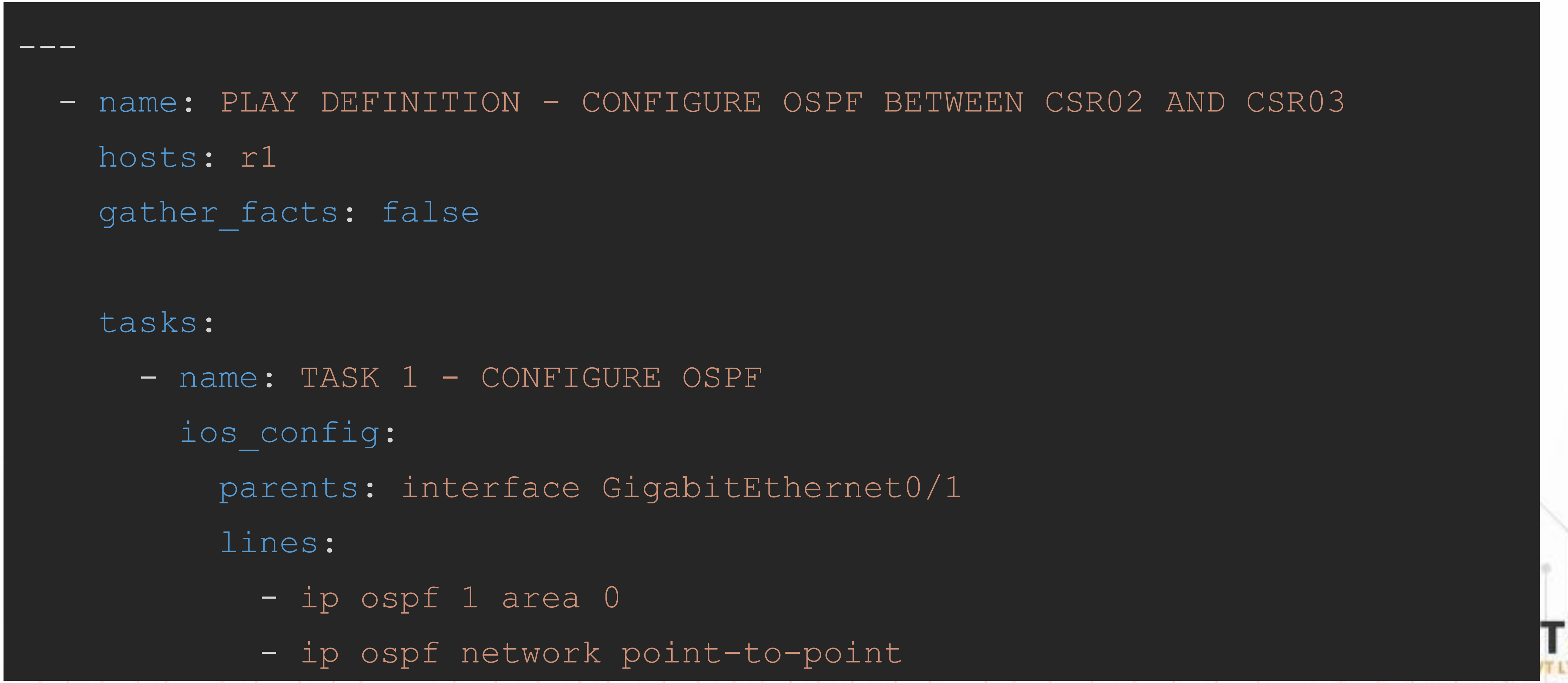


```
$ ansible-playbook playbooks/old/07ios_config.yml

PLAY [PLAY DEFINITION - DEPLOY SNMP COMMUNITY STRINGS ON IOS DEVICES]
*********************************************************************

TASK [TASK 1 - USE COMMANDS IN THE PLAYBOOK]
**********************************************************************************************
[WARNING]: To ensure idempotency and correct diff the input configuration lines should be
similar to how they appear if present in the
running configuration on device
changed: [r1]

PLAY RECAP
*************************************************************************************************
*******************************
r1                        : ok=1   changed=1        unreachable=0    failed=0 skipped=0
         rescued=0        ignored=0
```

IOS Configuration (Cont.)

- OSPF Configuration can be done as follows
- Since ospf config lines go under the interface, ios_config allows us to specify parent line under which the other lines need to be added

IOS Configuration (Cont.)



```
---
  - name: PLAY DEFINITION - CONFIGURE OSPF BETWEEN CSR02 AND CSR03
    hosts: r1
    gather_facts: false

    tasks:
      - name: TASK 1 - CONFIGURE OSPF
        ios_config:
          parents: interface GigabitEthernet0/1
          lines:
            - ip ospf 1 area 0
            - ip ospf network point-to-point
```

Ansible Check Mode

- Ansible provides a way to do a dry run before actually running the playbook against the devices
- We can use Ansible Check Mode for this
- We run the playbook command as it is just by adding a flag –check

```
$ ansible-playbook playbooks/ios_config_ospf.yml --check
```

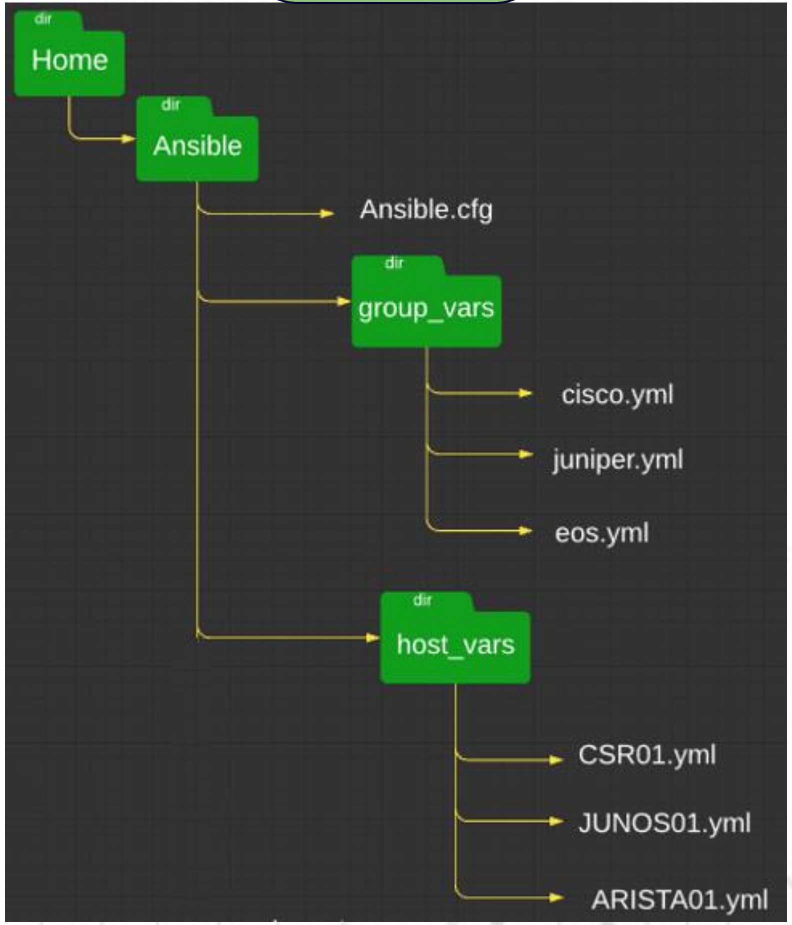
Host Variables

- Host variables can be defined in the inventory file or within a directory called host_vars.
- Variables that are specific to a host. It will be applicable to host only.
- Accessible within playbooks and templates.
- host_vars directory is the recommended location for host variables instead of specifying the variables inside the inventory file.

Not Recommended

```
#inventory
[switches]
10.1.1.1           ansible_port=22
switch1.cisco.com  ansible_user=apiuser

[routers]
r1.cisco.com   ansible_host=10.1.1.10 ansible_port=22
R2             ansible_ssh_pass=Nexaria@1
```

Recommended



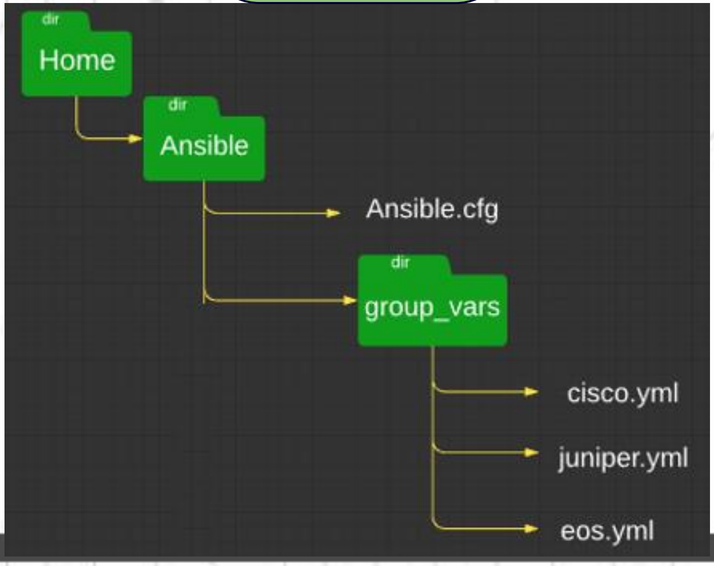
Group Variables

- Group variables can be defined in the inventory file or within a directory called group_vars
- Variables that are specific to a group. It will be applicable for all nodes that are part of that specific group
- Accessible within playbooks and templates
- Group_vars directory is the recommended location for group variables instead of specifying the variables inside the inventory file

Not Recommended

```
#inventory File

[routers]
r1.cisco.com
r2

[routers:vars]
snmp_ro=cisco
ansible_network_os=ios
```

Recommended



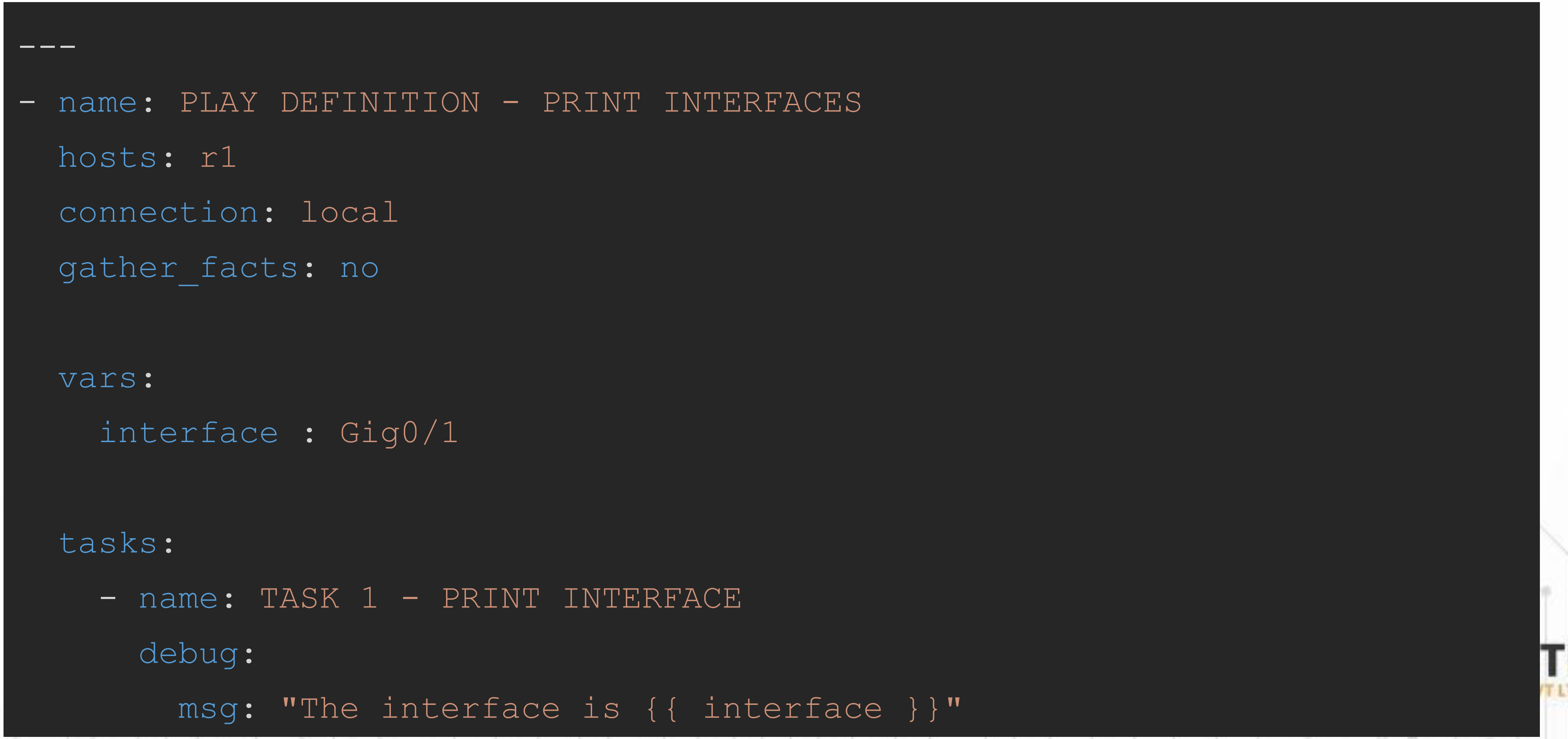
Verbosity in Ansible

- Verbosity levels are used to get the error information in detailed form.
- Verbosity also provides the facts information
- Use -v[vvv] to increase output verbosity
  - -v will show task results
  - -vv will show task results and task configuration
  - -vvv also shows information about connections to managed hosts
  - -vvvv adds information about plug-ins, users used to run scripts and names of scripts that are executed

Playbook Variables

- Ansible uses Jinja2 syntax for variables within a playbook, and uses curly brackets to indicate a variable e.g. {{ interface }}
- Variables within a playbook can be defined under the optional **vars** parameter.

Playbook Variables



```
---
- name: PLAY DEFINITION - PRINT INTERFACES
  hosts: r1
  connection: local
  gather_facts: no

  vars:
    interface : Gig0/1

  tasks:
    - name: TASK 1 - PRINT INTERFACE
      debug:
        msg: "The interface is {{ interface }}"
```

Playbook Variables - From File

- We can also load define variables in yaml files and load them using include_vars module

Playbook Variables - From File

```
---
- name: PLAY DEFINITION - PRINT INTERFACES
  hosts: r1
  connection: local
  gather_facts: no

  tasks:
    - name: load vars from file
      include_vars: vars/vars.yml

    - name: TASK 1 - PRINT INTERFACE
      debug:
        msg: "The interface is {{ interface }}"
```

Extra Variables

- Known as "extra vars"
- Variables passed into a playbook
- Highest priority

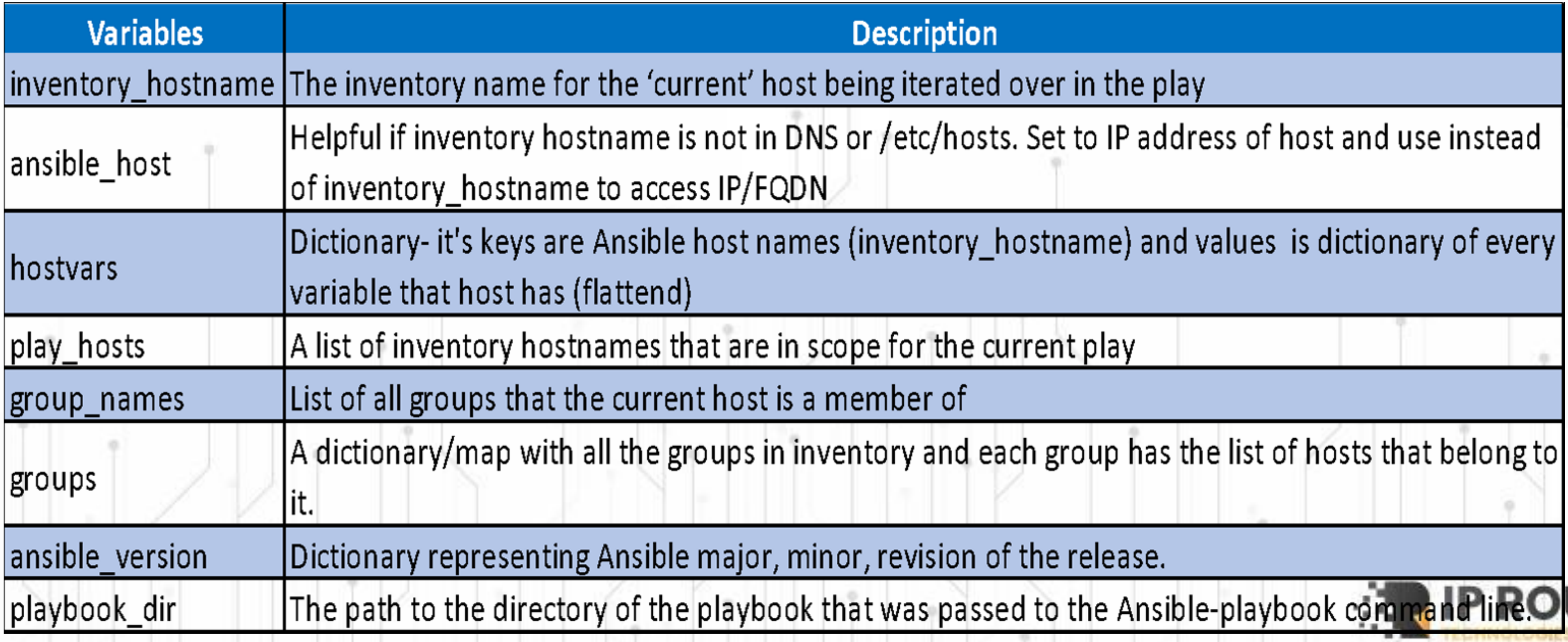
```
---
- name: display device clock
  hosts: "{{ devices }}"
  gather_facts: false

  tasks:
    - name: show clock on devices
      ios_command:
        commands: show clock
```

- Extra variables can be passed from the cli using –e or - - extra-vars flag

```
$ ansible-playbook playbook.yml -e “devices=r1”
$ ansible-playbook playbook.yml -e “devices=r1,r2”
$ ansible-playbook playbook.yml --extra-vars “devices=r3”
```

Special (Built-in) Variables

Ansible has several built in special variables.

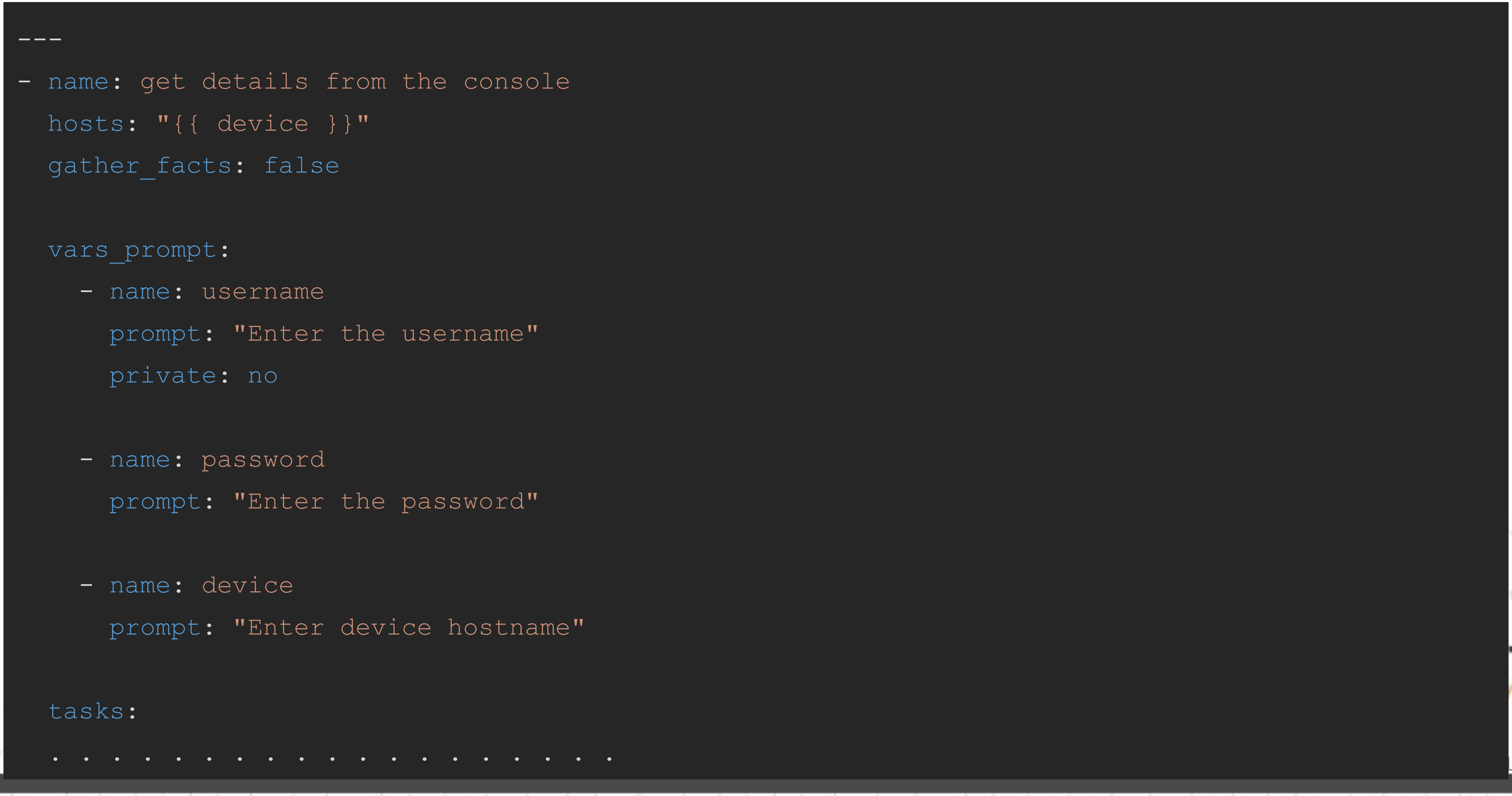


Variables	Description
inventory_hostname	The inventory name for the 'current' host being iterated over in the play
ansible_host	Helpful if inventory hostname is not in DNS or /etc/hosts. Set to IP address of host and use instead of inventory_hostname to access IP/FQDN
hostvars	Dictionary- it's keys are Ansible host names (inventory_hostname) and values is dictionary of every variable that host has (flattend)
play_hosts	A list of inventory hostnames that are in scope for the current play
group_names	List of all groups that the current host is a member of
groups	A dictionary/map with all the groups in inventory and each group has the list of hosts that belong to it.
ansible_version	Dictionary representing Ansible major, minor, revision of the release.
playbook_dir	The path to the directory of the playbook that was passed to the Ansible-playbook command line.

User Input

- At times we might need to accept input from the user and use it in playbook
- We can use vars_prompt to take input from the user

User Input



```yaml
---
- name: get details from the console
  hosts: "{{ device }}"
  gather_facts: false

  vars_prompt:
    - name: username
      prompt: "Enter the username"
      private: no

    - name: password
      prompt: "Enter the password"

    - name: device
      prompt: "Enter device hostname"

  tasks:
  . . . . . . . . . . . . . . . . . . .
```

Ansible Assertions

- Assertions can be used to check presence or absence of some text in the response
- This is useful when we want to use Ansible to ensure compliance
- e.g. We can write a playbook to fetch the version of a device and check the presence of an expected version in the output
    - If the response contains expected version, assertion is marked as pass else failure message is shown
- Ansible ***assert*** module can be used for this purpose

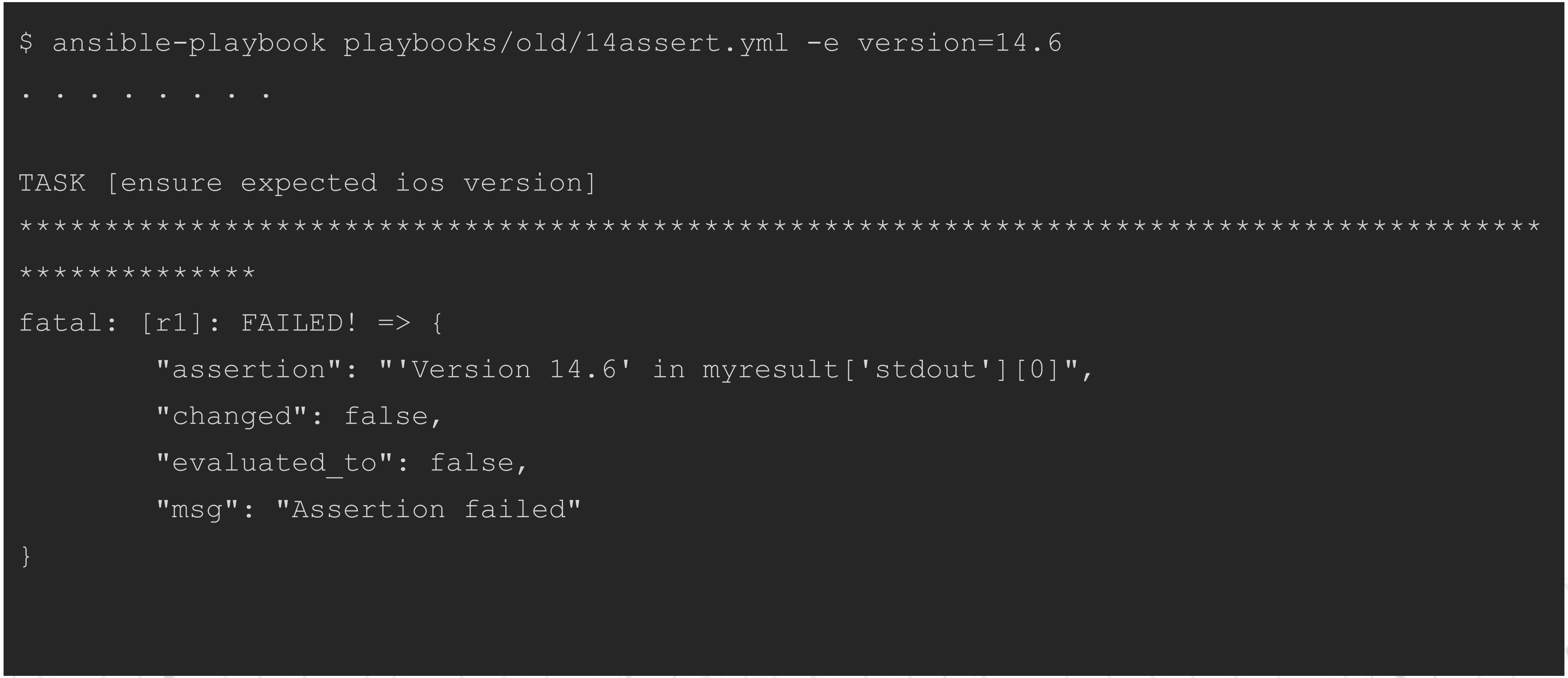
Ansible Assertions

```
---
- name: get version details and assert
  hosts: r1
  gather_facts: false

  tasks:
    - name: get version using show version
      ios_command:
        commands:
          - show version
      register: myresult

    - name: ensure expected ios version
      assert:
        that: "'Version {{ version }}' in myresult['stdout'][0]"
```

Ansible Assertions - Pass

```
$ ansible-playbook playbooks/old/14assert.yml  -e version=15.6
. . . . . . . .
TASK [get version using show version]
***********************************************************************************
ok: [r1]

TASK [ensure expected ios version]
***********************************************************************************
ok: [r1] => {
        "changed": false,
        "msg": "All assertions passed"
}

PLAY RECAP
***********************************************************************************
r1                      : ok=2   changed=0       unreachable=0   failed=0        skipped=0
        rescued=0       ignored=0
```

Ansible Assertions - Fail



```
$ ansible-playbook playbooks/old/14assert.yml -e version=14.6
. . . . . . . .

TASK [ensure expected ios version]
*****************************************************************************************
**************
fatal: [r1]: FAILED! => {
        "assertion": "'Version 14.6' in myresult['stdout'][0]",
        "changed": false,
        "evaluated_to": false,
        "msg": "Assertion failed"
}
```

Ansible Loops

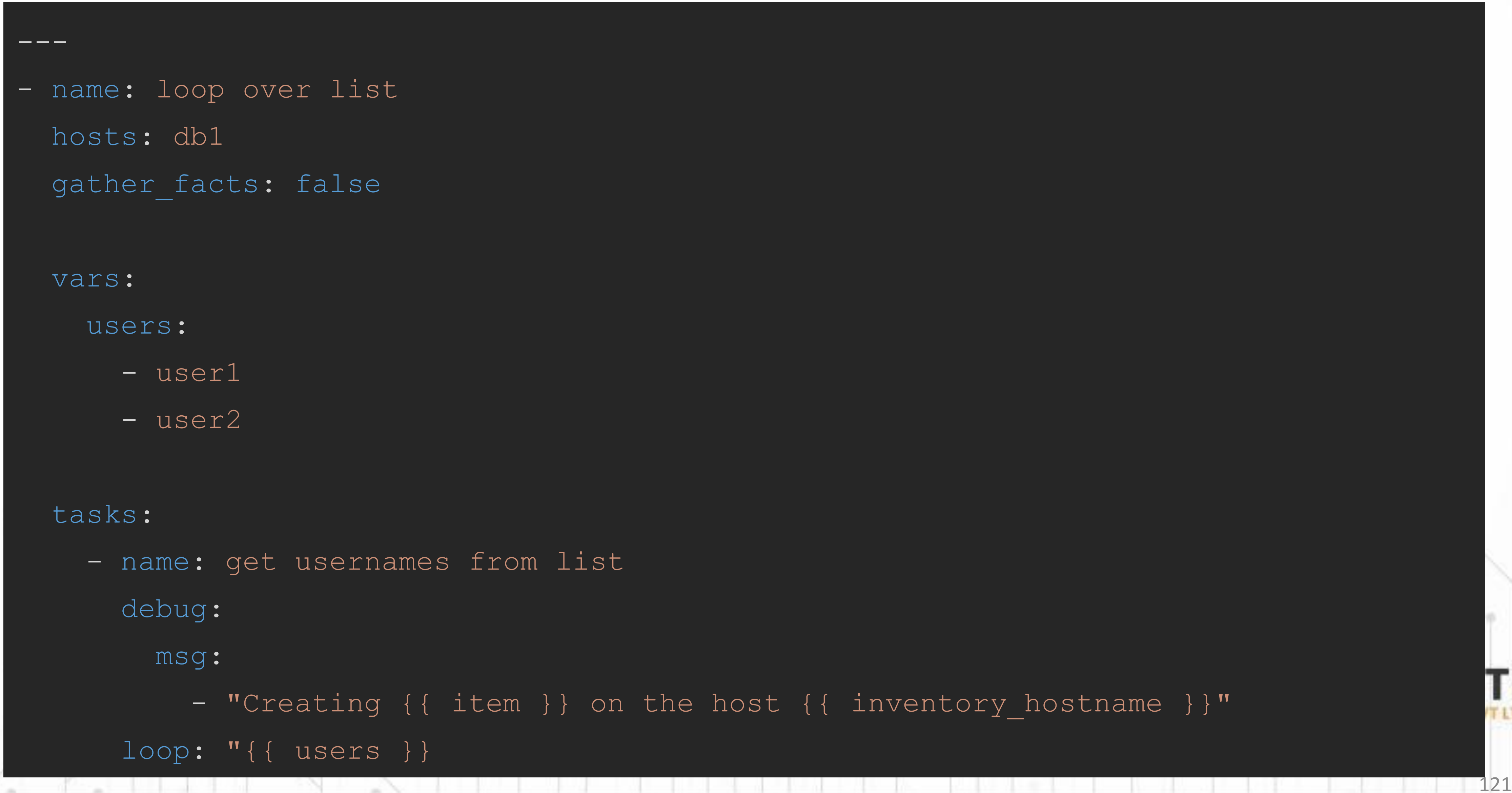
Loops are a programming element that repeat a portion of code a set number of times until the desired process is complete.

Repetitive tasks are common in programming, and loops are essential to save time and minimize errors.

In An Ansible we can iterate over :

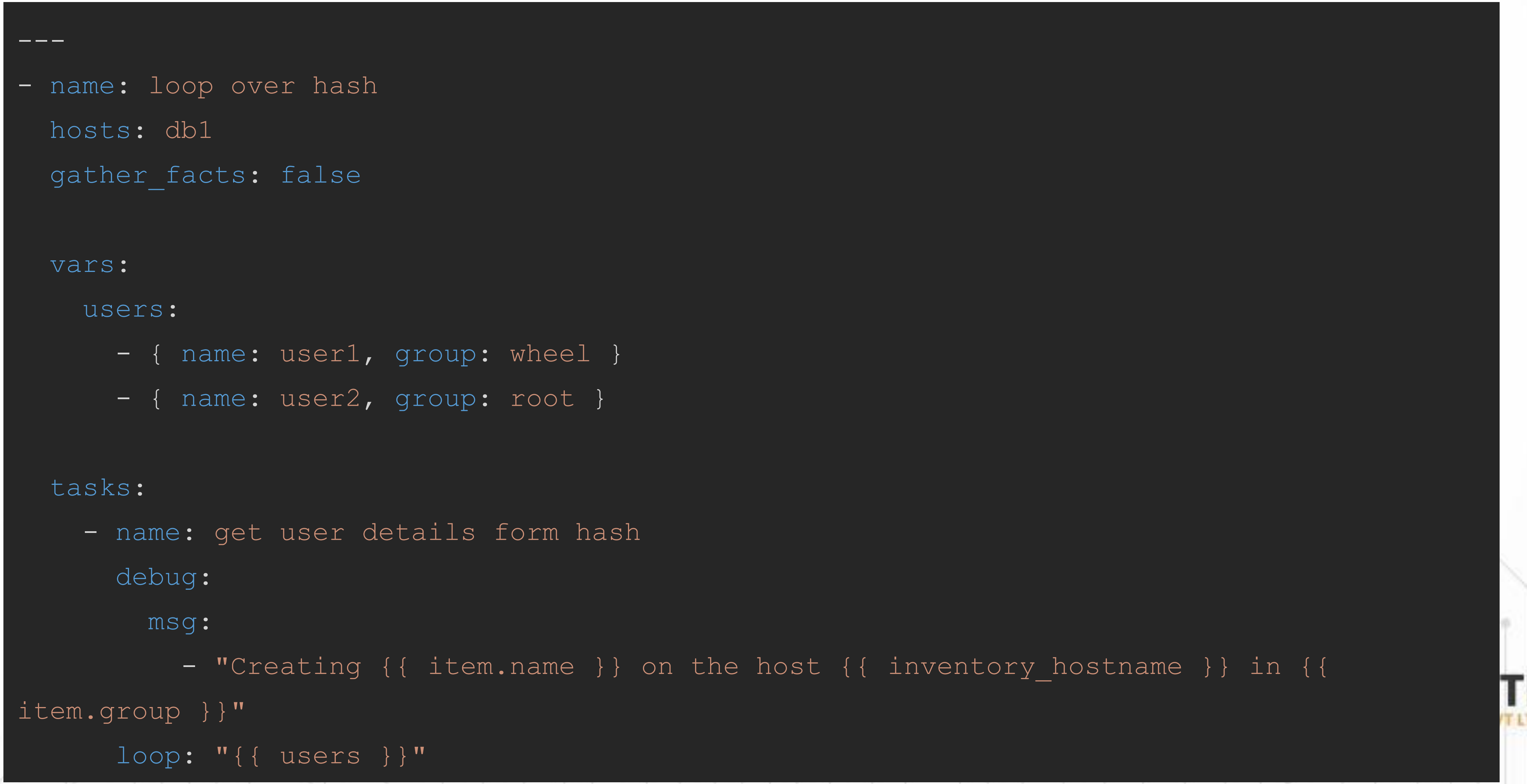
1) List
2) List of hashes
3) Dictionary
4) Nested lists

Looping Over Ansible List



```
---
- name: loop over list
  hosts: db1
  gather_facts: false

  vars:
    users:
      - user1
      - user2

  tasks:
    - name: get usernames from list
      debug:
        msg:
          - "Creating {{ item }} on the host {{ inventory_hostname }}"
      loop: "{{ users }}
```

Looping Over Ansible Hash (JSON Obj)



```
---
- name: loop over hash
  hosts: db1
  gather_facts: false

  vars:
    users:
      - { name: user1, group: wheel }
      - { name: user2, group: root }

  tasks:
    - name: get user details form hash
      debug:
        msg:
          - "Creating {{ item.name }} on the host {{ inventory_hostname }} in {{ item.group }}"
      loop: "{{ users }}"
```

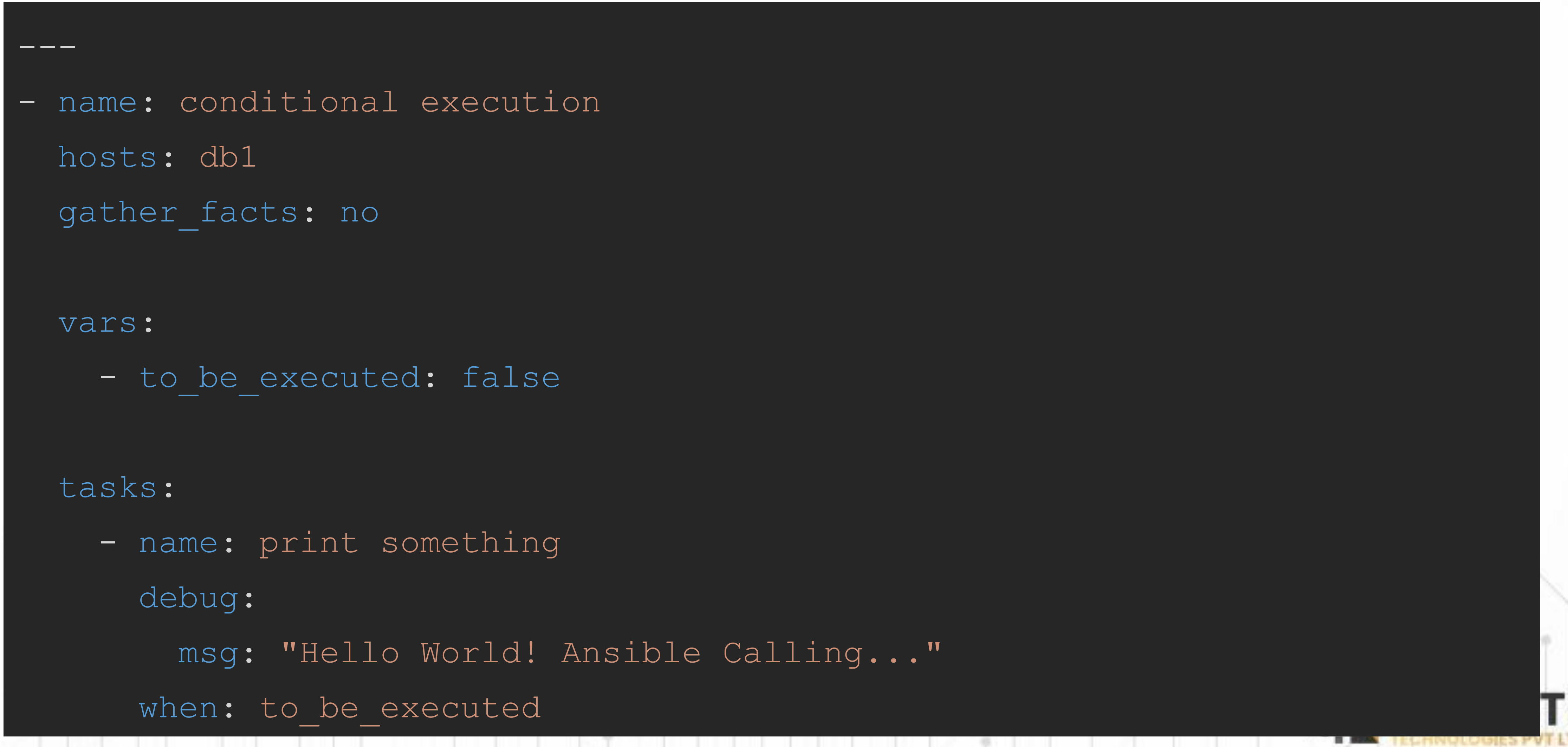
Looping Over Ansible Dictionary

```yaml
---
- name: get version details and assert
  hosts: db1
  gather_facts: false

  vars:
    users:
      - name: user1
        group: wheel
      - name: user2
        group: root

  tasks:
    - name: get version using show version
      debug:
        msg:
          - "Creating {{ item.name }} on the host {{ inventory_hostname }} in {{ item.group }}"
      loop: "{{ users }}"
```

Ansible Conditionals

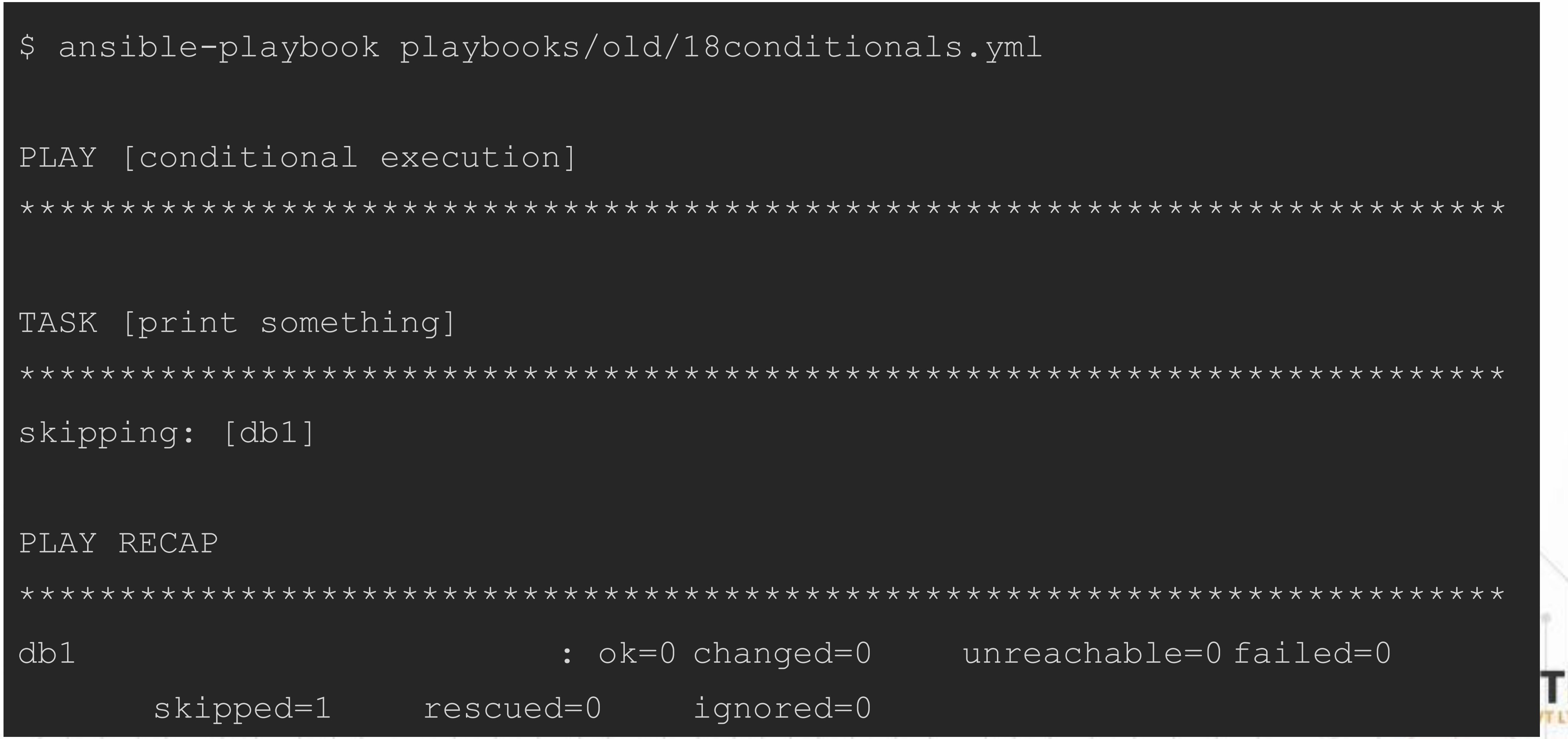
- Ansible can use conditionals to execute tasks or plays when certain conditions are met
- For example, a conditional can be used to determine available memory on a managed host before Ansible installs or configures a service
- Help differentiate between managed hosts and assign them functional roles based on the conditions
- Playbook variables, registered variables, and Ansible facts can all be tested with conditionals
- Operators to compare strings, numeric data, and Boolean values are available

Ansible Conditionals (Cont.)



```yaml
---
- name: conditional execution
  hosts: db1
  gather_facts: no

  vars:
    - to_be_executed: false

  tasks:
    - name: print something
      debug:
        msg: "Hello World! Ansible Calling..."
      when: to_be_executed
```

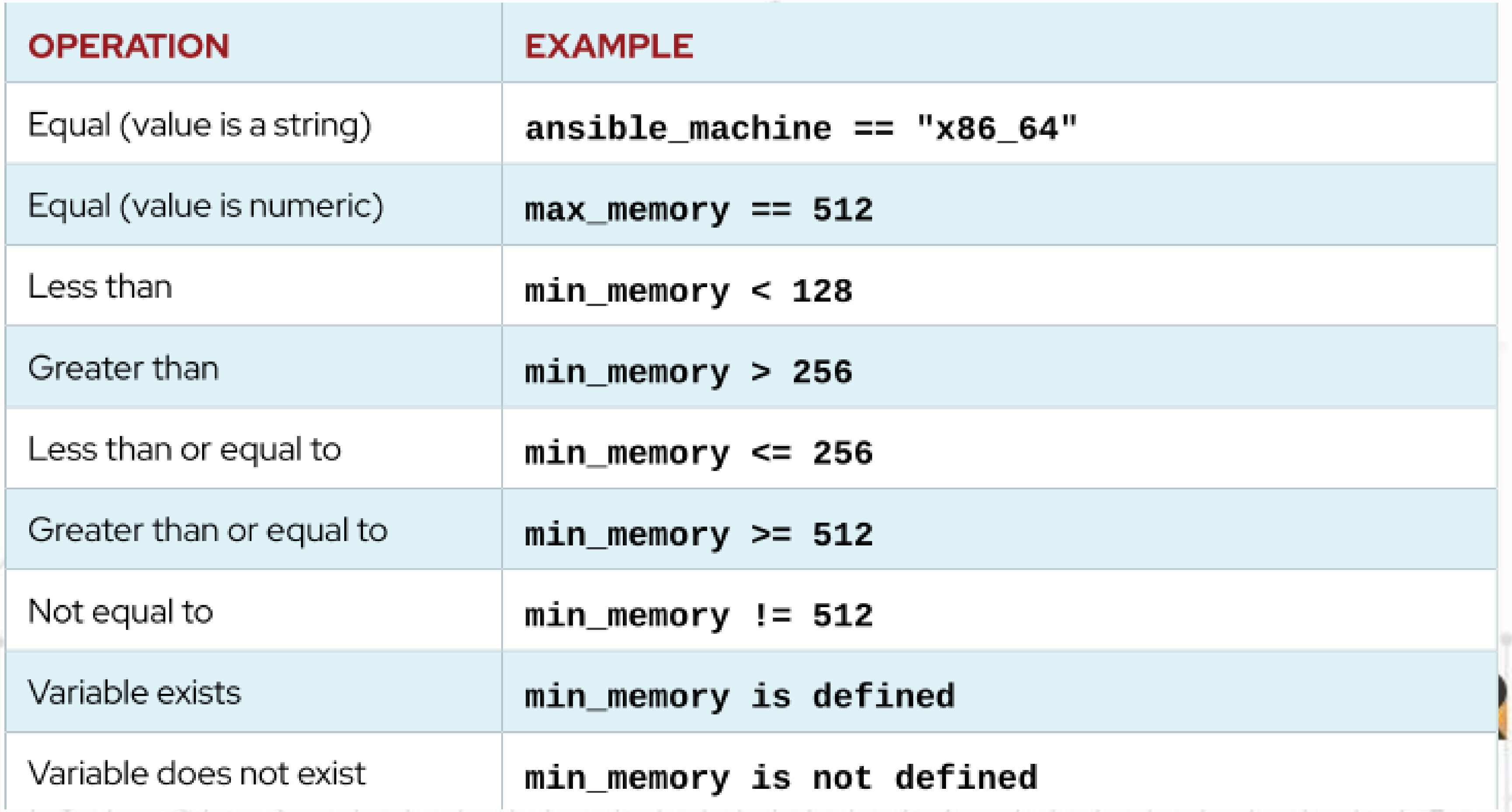
Ansible Conditionals (Cont.)



```
$ ansible-playbook playbooks/old/18conditionals.yml

PLAY [conditional execution]
***************************************************************************

TASK [print something]
***************************************************************************
skipping: [db1]

PLAY RECAP
***************************************************************************
db1                        : ok=0 changed=0    unreachable=0 failed=0
       skipped=1    rescued=0    ignored=0
```

Running Tasks Conditionally (Cont.)

- Defining conditions using operations is shown below



OPERATION	EXAMPLE
Equal (value is a string)	`ansible_machine == "x86_64"`
Equal (value is numeric)	`max_memory == 512`
Less than	`min_memory < 128`
Greater than	`min_memory > 256`
Less than or equal to	`min_memory <= 256`
Greater than or equal to	`min_memory >= 512`
Not equal to	`min_memory != 512`
Variable exists	`min_memory is defined`
Variable does not exist	`min_memory is not defined`

Ansible Vault

- Ansible needs sensitive data such as passwords or API keys to configure managed hosts
- Normally stored in playbooks or other files in vars as plain text
- When playbooks stored in GITHub for version management or back up this is a Security risk and policy violation
- Ansible Vault can be used to encrypt / decrypt such files
- Comes bundled with Ansible install
- Available as command line tool called ansible-vault
- Can be used to create, edit, encrypt, decrypt and view files containing sensitive info

Ansible Vault (Cont.)

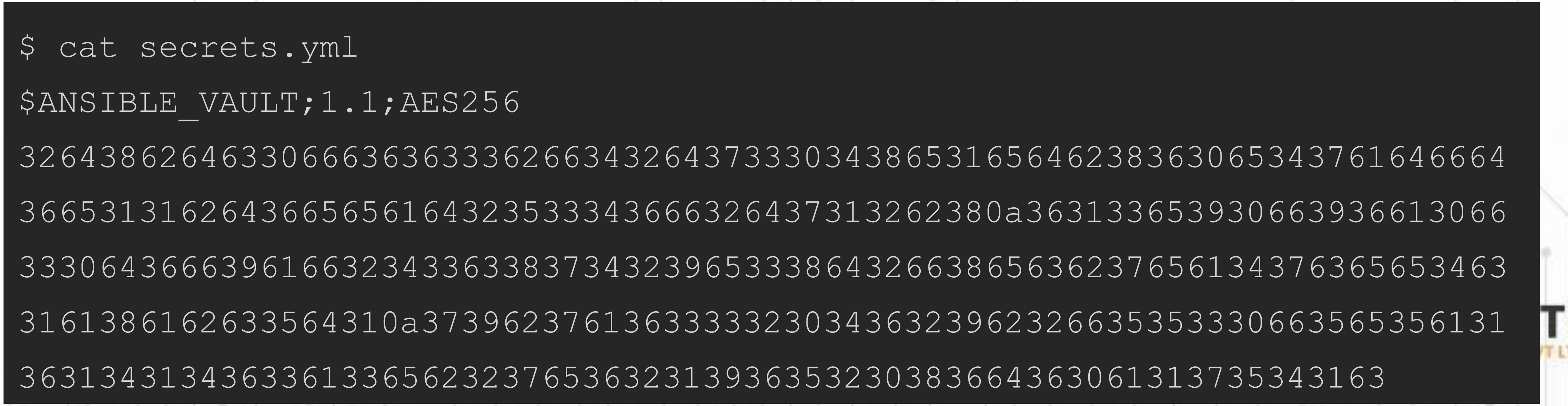
- Creating secrets file by entering the password directly
- Once password and confirm password are entered default vi editor is opened
- We can change this by setting an env variable EDITOR
  - export EDITOR=nano

```
$ ansible-vault create mysecrets.yml
New Vault password:
Confirm New Vault password:
```

```
$ export EDITOR=nano
```

Ansible Vault (Cont.)

- Edit the file either in vi or nano and put some variable
  - ansible_password=cisco
- Encrypted file is created successfully
- Try to open this file directly using cat or vi or nano



```
$ cat secrets.yml
$ANSIBLE_VAULT;1.1;AES256
32643862646330666363633362663432643733303438653165646238363065343761646664
3665313162643665656164323533343666326437313262380a363133653930663936613066
3330643666396166323433633837343239653338643266386563623765613437636565346 3
31613861626335643 10a373962376136333332303436323962326635353330663565356131
36313431343633613365623237653632313936353230383664363061313735343163
```

Ansible Vault (Cont.)

- Use ***ansible-vault view*** to view the encrypted file

```
$ ansible-vault view secrets.yml
Vault password:
ansible_password=cisco
```

- Use ***ansible-vault edit*** to edit the encrypted file

```
$ ansible-vault edit secrets.yml
Vault password:
```

```
  GNU nano 4.8                    /home/nexadmin/.ansible/tmp/ansible-
local-16690vhugg19d/tmpfh9qmirz.yml
ansible_password=cisco
```

Ansible Vault (Cont.)

- Use ***ansible-vault decrypt*** to decrypt the encrypted file

```
$ ansible-vault decrypt secrets.yml
Vault password:
Decryption successful
$ cat secrets.yml
ansible_password=cisco
```

Ansible Vault (Cont.)

- Use ***ansible-vault encrypt*** to encrypt an unencrypted file

```
$ ansible-vault encrypt secrets.yml
New Vault password:
Confirm New Vault password:
Encryption successful
```

Ansible Vault (Cont.)

- Use ***ansible-vault rekey*** to change the encryption password

```
$ ansible-vault rekey secrets.yml
Vault password:
New Vault password:
Confirm New Vault password:
Rekey successful
```

Ansible Vault (Cont.)

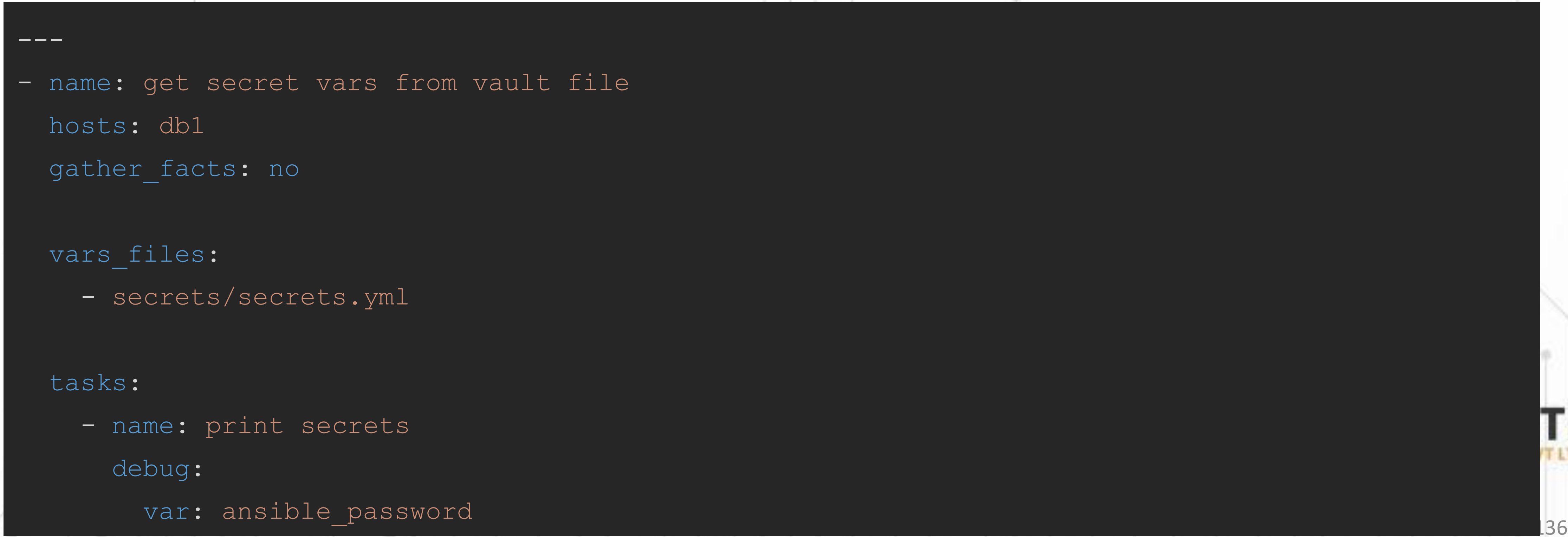
- Instead of entering the encryption password on command line, we can put it in a file and use the file for all the vault operations

```
$ nano vault-pass
$ cat vault-pass
devnet
```

```
$ ansible-vault create secrets.yml --vault-password-file=vault-pass
$ ansible-vault view secrets.yml --vault-password-file=vault-pass
ansible_password=cisco
$ ansible-vault decrypt secrets.yml --vault-password-file=vault-pass
Decryption successful
$ ansible-vault encrypt secrets.yml --vault-password-file=vault-pass
Encryption successful
```

Using Vault

- We are now ready to take advantage of running playbooks securely reading sensitive data like passwords, api keys etc from vault encrypted files
- Write a playbooks that uses something from the encrypted file



```
---
- name: get secret vars from vault file
  hosts: db1
  gather_facts: no

  vars_files:
    - secrets/secrets.yml

  tasks:
    - name: print secrets
      debug:
        var: ansible_password
```

Using Vault (Cont.)

- Try running the playbook in the usual way

```
$ ansible-playbook playbooks/old/18vault_get_pwd.yml
ERROR! Attempting to decrypt but no vault secrets found
```

- Now try running the playbook with the vault password
- –vault-id is the flag used to send the password to decrypt the secret file
- @prompt allows user to enter this password at CLI prompt

```
$ ansible-playbook playbooks/old/18vault_get_pwd.yml --vault-id @prompt
Vault password (default):
PLAY [get secret vars from vault file]
************************************************************************************
TASK [print secrets]
*********************************************************************************************
ok: [db1] => {
        "ansible_password": "cisco"
}
```

Using Vault (Cont.)

- We can prevent the password being prompted and let Ansible read it from password file
- This is similar to the way we used password file for vault operations

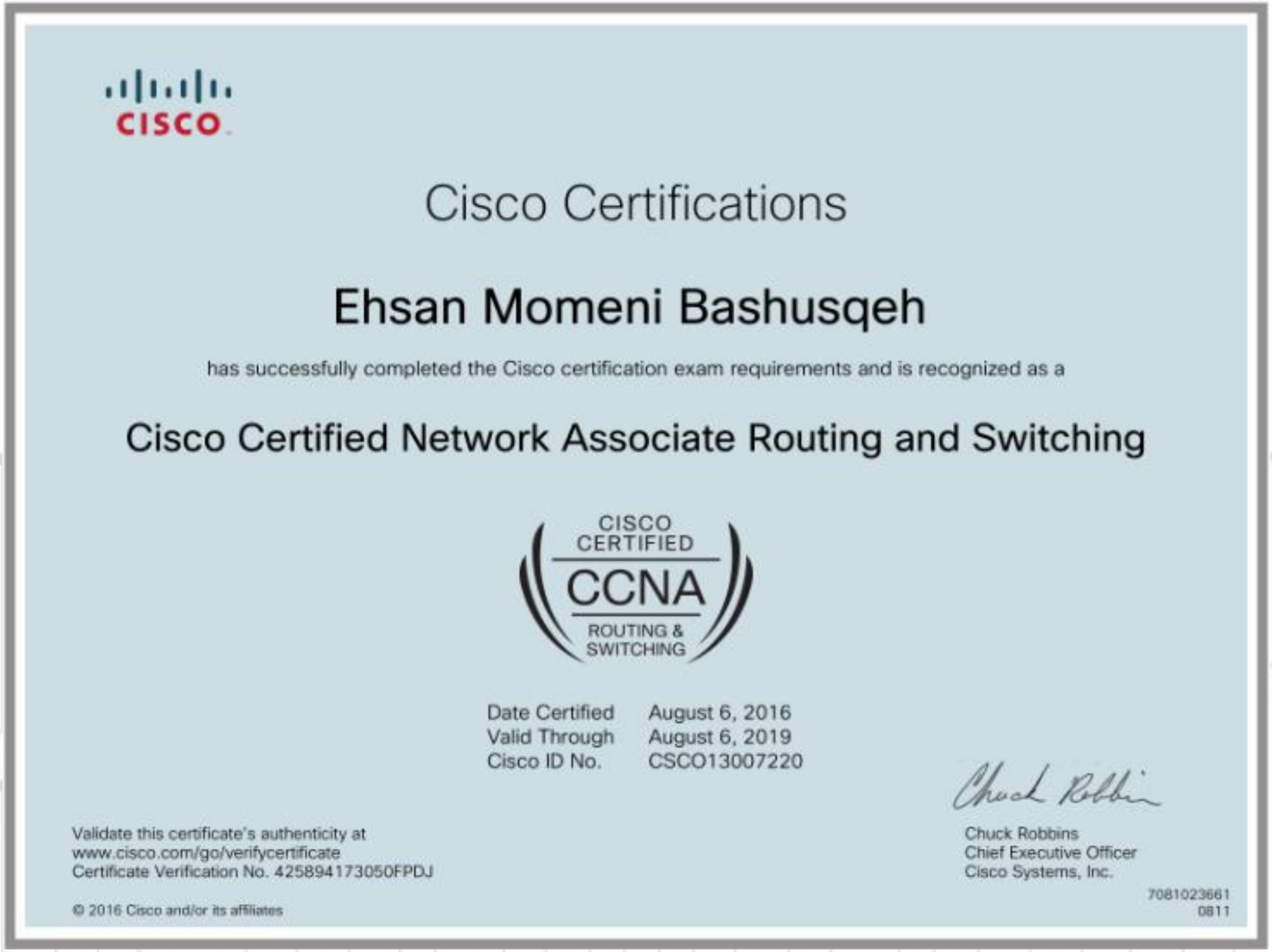
```
$ ansible-playbook playbooks/old/18vault$ ansible-playbook
playbooks/old/18vault_get_pwd.yml --vault-password-file=playbooks/old/secrets/vault-pass

PLAY [get secret vars from vault file]
***********************************************************************************


TASK [print secrets]
**************************************************************************************
ok: [db1] => {
        "ansible_password": "cisco"
}
```

Introduction

Jinja2 is a feature rich templating language widely used in the Python ecosystem.

- Can be used directly in Python programs
- Can also be used in a wide range of applications as their template rendering engine
- e.g.
  - Web frameworks like Django, Flask etc.
  - Configuration management tools like Ansible, Saltstack
  - Static site generator tools like Pelican and so on......
- Let’s try to put things in perspective......

Sample Cisco Certificate



Cisco Certifications

Ehsan Momeni Bashusqeh

has successfully completed the Cisco certification exam requirements and is recognized as a

Cisco Certified Network Associate Routing and Switching

CISCO CERTIFIED
CCNA
ROUTING & SWITCHING

Date Certified	August 6, 2016
Valid Through	August 6, 2019
Cisco ID No.	CSCO13007220

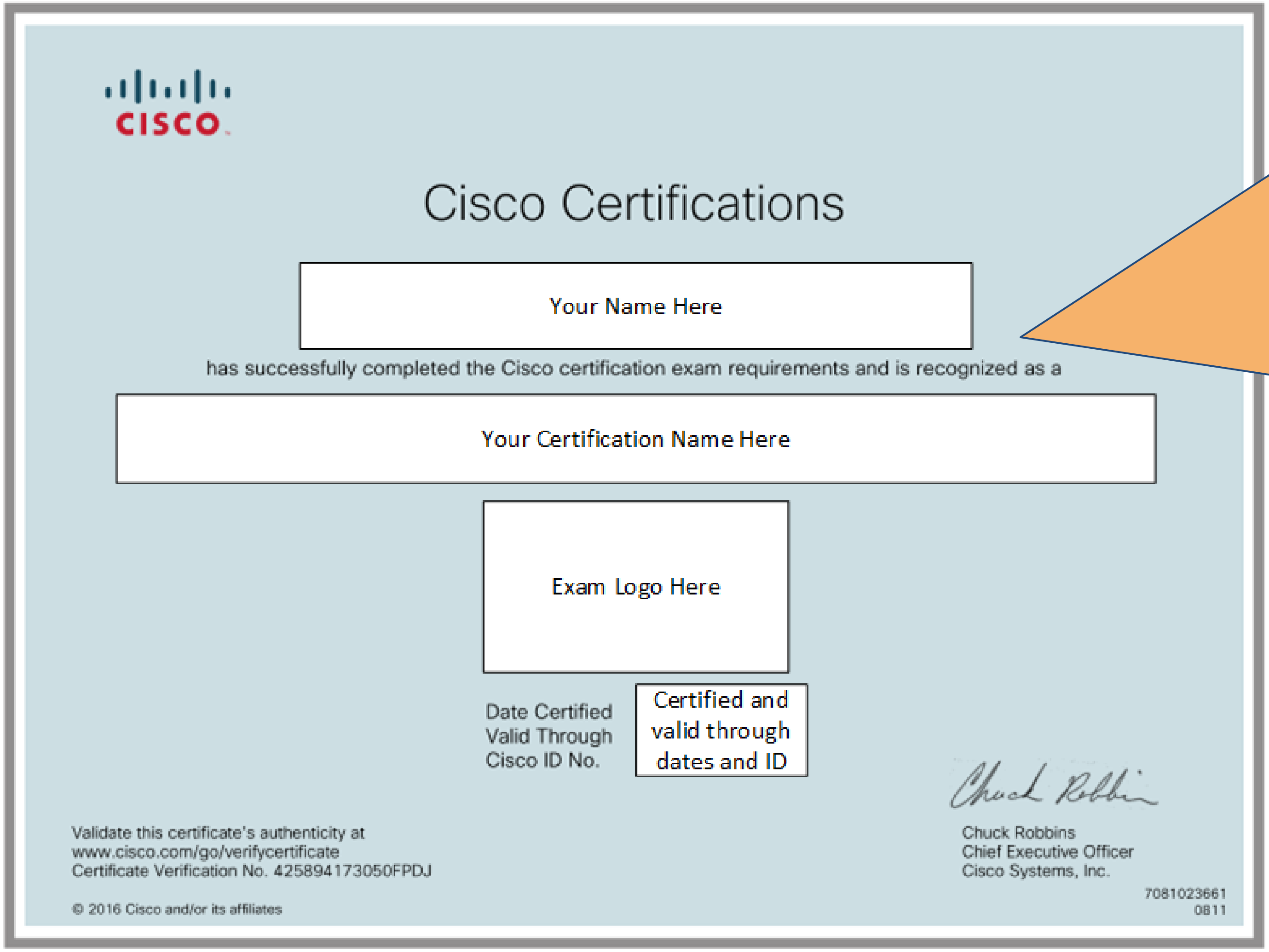
Chuck Robbins

Chuck Robbins
Chief Executive Officer
Cisco Systems, Inc.

Validate this certificate's authenticity at
www.cisco.com/go/verifycertificate
Certificate Verification No. 425894173050FPDJ

7081023661
0811

Template



Cisco Certifications

Your Name Here

has successfully completed the Cisco certification exam requirements and is recognized as a

Your Certification Name Here

Exam Logo Here

Date Certified
Valid Through
Cisco ID No.

Certified and valid through dates and ID

Validate this certificate's authenticity at
www.cisco.com/go/verifycertificate
Certificate Verification No. 425894173050FPDJ

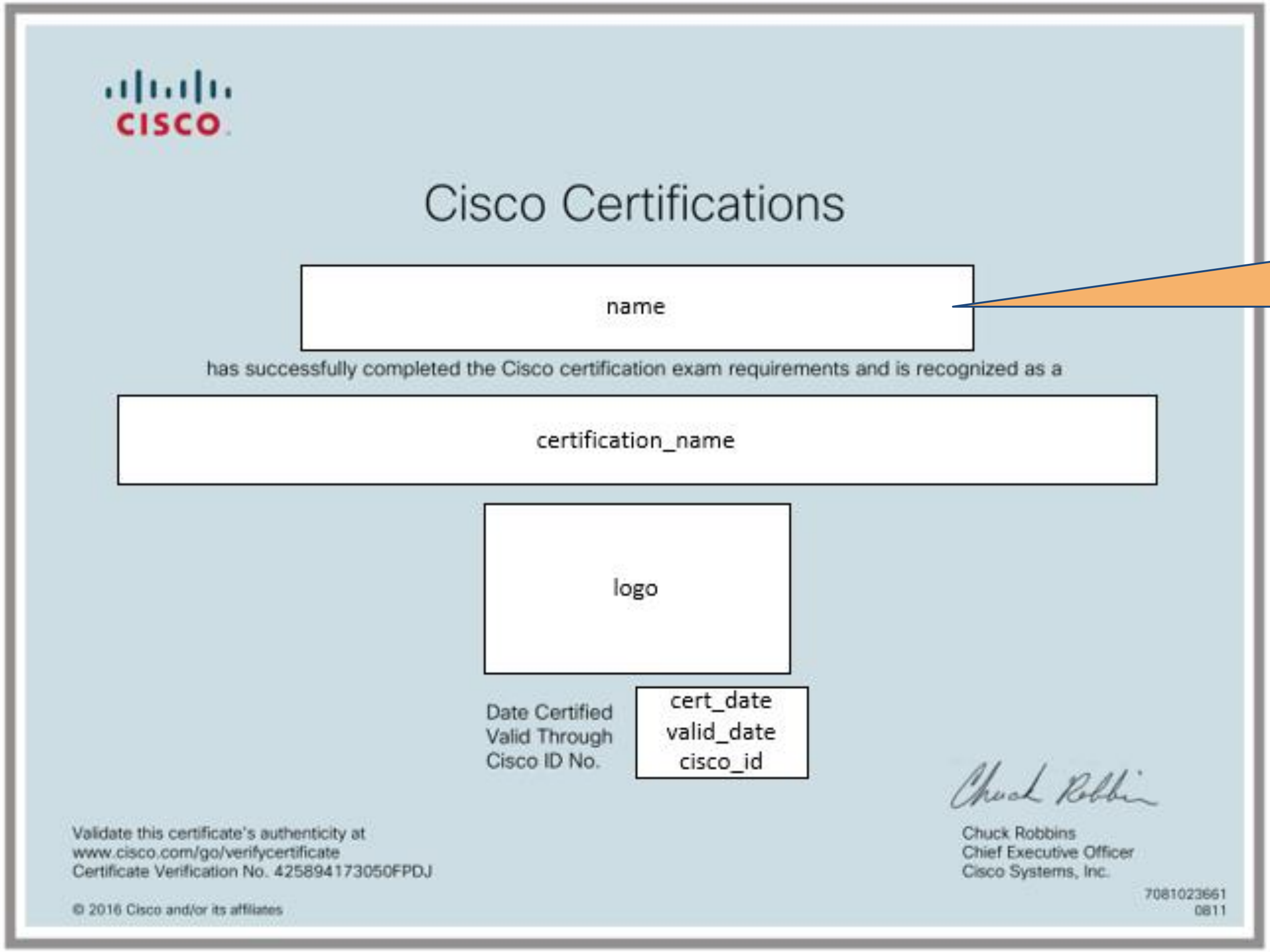
Chuck Robbins
Chief Executive Officer
Cisco Systems, Inc.

7081023661
0811

© 2016 Cisco and/or its affiliates

- Imagine Cisco having to print thousands of such certificates for the vast number of certifications they offer
- They create a template with placeholders for the name of the person, certification, logo, dates etc.
- The things that tend to change from one certificate to the other
- The fixed part, along with placeholders becomes a template

Template with variable names



Cisco Certifications

name

has successfully completed the Cisco certification exam requirements and is recognized as a

certification_name

logo

Date Certified	cert_date
Valid Through	valid_date
Cisco ID No.	cisco_id

Chuck Robbins
Chief Executive Officer
Cisco Systems, Inc.

Validate this certificate's authenticity at
www.cisco.com/go/verifycertificate
Certificate Verification No. 425894173050FPDJ

© 2016 Cisco and/or its affiliates

7081023661
0811

Placeholders in computer terms are nothing but variables

Data stored in variables

name

Ehsan Momeni Bashusqeh

certification_name

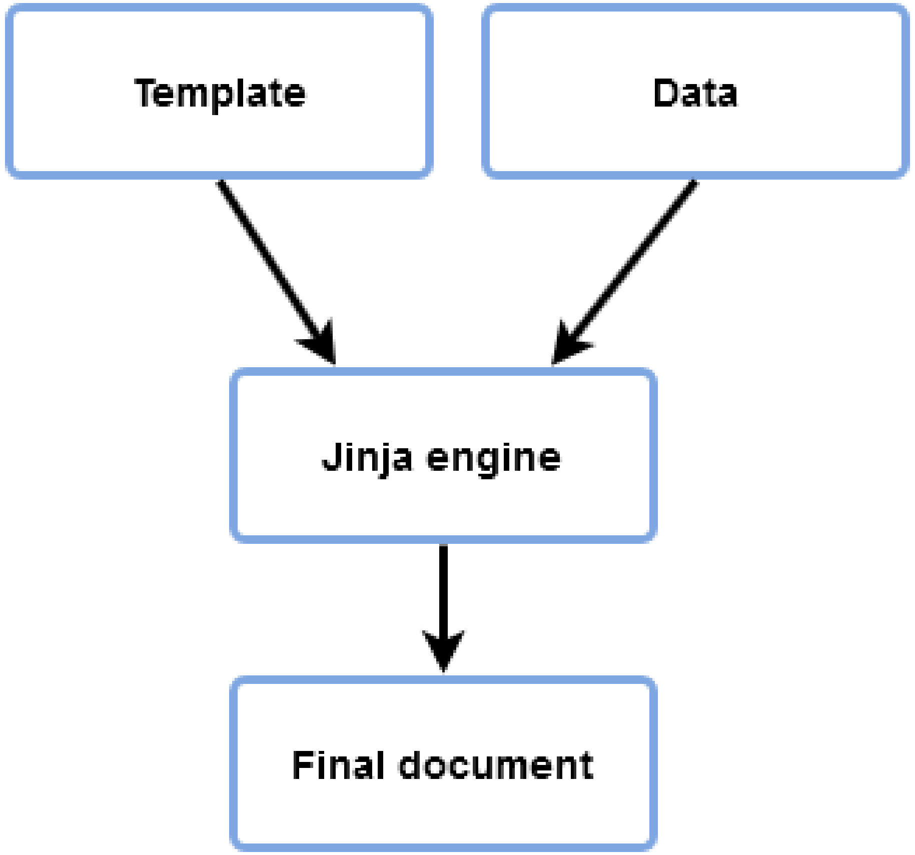
Cisco Certified Network Associate Routing and Switching

logo

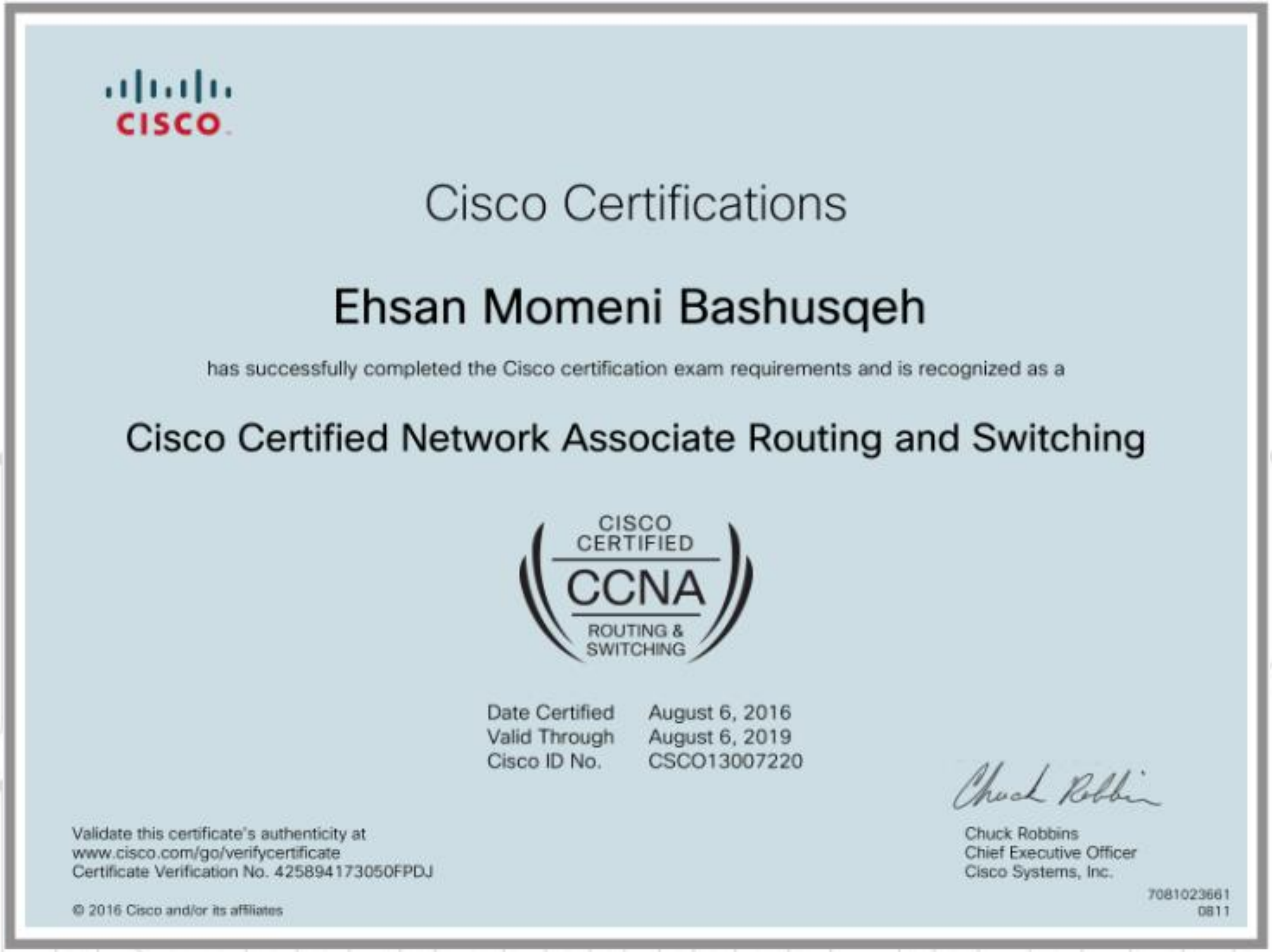
cert_date
valid_date
cisco_id

August 6, 2016
August 6, 2019
CSCO13007220

Enters Jinja2



Final Document



CISCO

Cisco Certifications

Ehsan Momeni Bashusqeh

has successfully completed the Cisco certification exam requirements and is recognized as a

Cisco Certified Network Associate Routing and Switching

CISCO
CERTIFIED
CCNA
ROUTING &
SWITCHING

Date Certified	August 6, 2016
Valid Through	August 6, 2019
Cisco ID No.	CSCO13007220

Validate this certificate's authenticity at
www.cisco.com/go/verifycertificate
Certificate Verification No. 425894173050FPDJ

Chuck Robbin

Chuck Robbins
Chief Executive Officer
Cisco Systems, Inc.

7081023661
0811

Ansible with Jinja2

- As we know Ansible has a large collection of modules, Jinja2 module also comes preinstalled with Ansible
- Ansible provides variables to the templates and renders them using the ***template*** module which in turn calls the rendering engine of Jinja2
- Template rendering happens on Ansible controller
- Rendered task is then sent to the target machine for execution
- This is done to minimize the package requirement on target machine
- This also limits the amount of data Ansible passes to the target machine

Jinja2 Templating

- Jinja2 needs the following source ingredients to work
  - Template
  - Data
- Data can come from various sources like
  - JSON data returned by API
  - Loaded from static YAML file
  - Python dictionary defined in our application
- Basic idea is to identify static and dynamic parts of the documents
- Dynamic parts are parametrized, so they change according to the data passed
- Hence multiple versions of the document are created with static part being the same and dynamic part changing as per the data passed

A more relevant use case - BGP Configuration

```
router bgp 45000
 router-id 172.17.1.99
 bgp log-neighbor-changes
 neighbor 192.168.1.2 remote-as 40000
 neighbor 192.168.3.2 remote-as 50000
 address-family ipv4 unicast
  neighbor 192.168.1.2 activate
  network 172.17.1.0 mask 255.255.255.0
  exit-address-family
```

Sample target config we want to generate

- Shown above is a short snippet of CIsco IOS configuration
- First we identify which part of the above snippet is static and which parts change between devices
- Typically ASNs, IP Addresses, address family type etc. change between the devices
- The parts that change are converted into variables to be substituted with actual data when template is rendered at runtime

A more relevant use case - BGP Configuration

```
router bgp {{ local_asn }}
 router-id {{ router_id }}
 bgp log-neighbor-changes
 neighbor {{ neighbor_id_1 }} remote-as {{ remote_asn_1 }}
 neighbor {{ neighbor_id_2 }} remote-as {{ remote_asn_2 }}
 address-family ipv4 unicast
  neighbor {{ neighbor_id_1 }} activate
  network {{ network }} mask {{ net_mask }}
  exit-address-family
```

Actual values replaced with variables. This becomes a template now.

- In Jinja2 anything found between a pair of double opening and closing curly braces (“{{”, “}}”), known as delimiters, will be evaluated and replaced by the engine
- The templating engine expects to find a variable with the same name in the list of variables
- The variable name in the template will then be replaced with the value from the data file which can be a JSON file, YAML file, Python dictionary etc.

A more relevant use case - BGP Configuration

Variable Set 1

```
local_asn: 45000
router_id: 172.17.1.99
neighbor_id_1: 192.168.1.2
neighbor_id_2: 192.168.3.2
remote_asn_1: 40000
remote_asn_2: 50000
network: 172.17.1.0
net_mask: 255.255.255.0
```

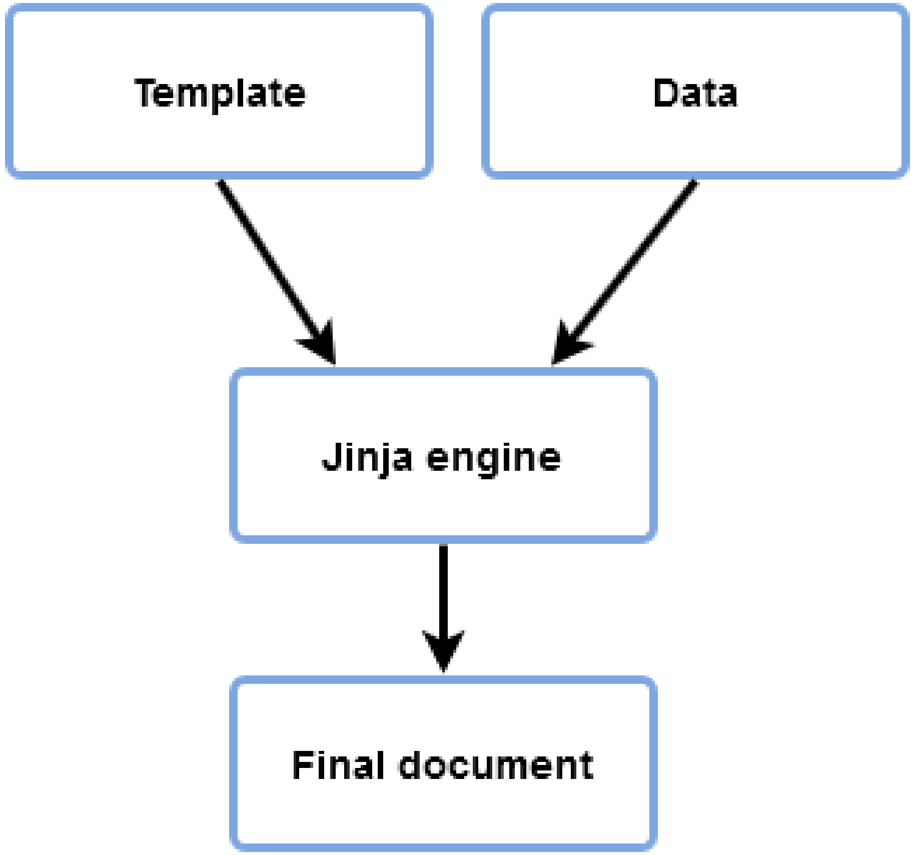
- When we substitute Variable Set 1 into the template, we get first set of BGP config commands
- Similarly when we substitute Variable Set 2 into the template, we get second set of BGP config commands

Variable Set 2

```
local_asn: 95000
router_id: 172.17.1.200
neighbor_id_1: 192.168.10.200
neighbor_id_2: 192.168.20.200
remote_asn_1: 70000
remote_asn_2: 80000
network: 172.17.1.0
net_mask: 255.255.255.0
```

IP ROUTE
TECHNOLOGIES PVT LTD

Enters Jinja2



A more relevant use case - BGP Configuration

Target config 1

```
router bgp 45000
 router-id 172.17.1.99
 bgp log-neighbor-changes
 neighbor 192.168.1.2 remote-as 40000
 neighbor 192.168.3.2 remote-as 50000
 address-family ipv4 unicast
  neighbor 192.168.1.2 activate
  network 172.17.1.0 mask 255.255.255.0
  exit-address-family
```

- So you get the idea
- You pass 2 sets of data to get 2 sets of config
- Pass ‘n’ sets of data to get ‘n’ sets of config

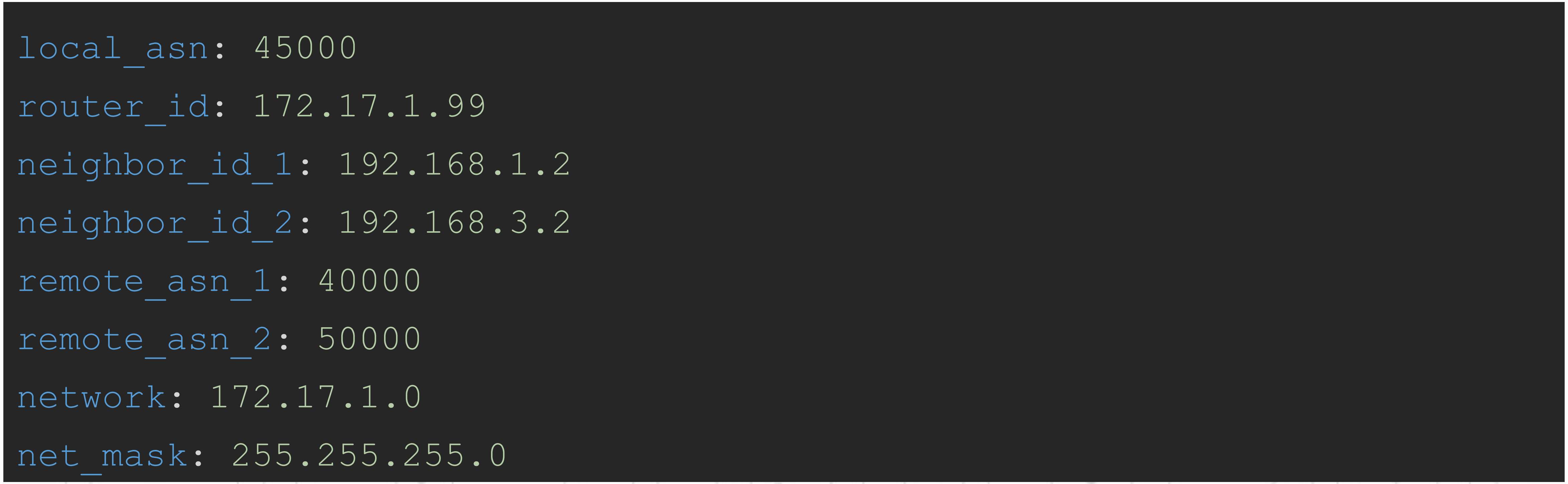
Target config 2

```
router bgp 95000
 router-id 172.17.1.200
 bgp log-neighbor-changes
 neighbor 192.168.10.200 remote-as 70000
 neighbor 192.168.20.200 remote-as 80000
 address-family ipv4 unicast
  neighbor 192.168.10.200 activate
  network 172.17.1.0 mask 255.255.255.0
  exit-address-family
```

Jinja2 Templating - Template

```
router bgp {{ local_asn }}
 router-id {{ router_id }}
 bgp log-neighbor-changes
 neighbor {{ neighbor_id_1 }} remote-as {{ remote_asn_1 }}
 neighbor {{ neighbor_id_2 }} remote-as {{ remote_asn_2 }}
 address-family ipv4 unicast
  neighbor {{ neighbor_id_1 }} activate
  network {{ network }} mask {{ net_mask }}
  exit-address-family
```

Jinja2 Templating - Variable File



```
local_asn: 45000
router_id: 172.17.1.99
neighbor_id_1: 192.168.1.2
neighbor_id_2: 192.168.3.2
remote_asn_1: 40000
remote_asn_2: 50000
network: 172.17.1.0
net_mask: 255.255.255.0
```

Jinja2 Templating - Playbook

```yaml
---
- name: bgp config generation using jinja2
  hosts: r1
  gather_facts: no

  vars_files:
    - vars/bgp_config_vars.yml

  tasks:
    - name: generate bgp config using templates and variables
      template:
        src: jinja2/templates/bgp_config.j2
        dest: jinja2/dest/bgp_config.txt
      register: output
      delegate_to: localhost
```

Jinja2 Templating - Result File

```
router bgp 45000
 router-id 172.17.1.99
 bgp log-neighbor-changes
 neighbor 192.168.1.2 remote-as 40000
 neighbor 192.168.3.2 remote-as 50000
 address-family ipv4 unicast
  neighbor 192.168.1.2 activate
  network 172.17.1.0 mask 255.255.255.0
  exit-address-family
```

Software Version Control

With Git

Software Version Control

- Process of saving various copies of a file or set of files in order to track changes made to those files
- Involves a database that stores current and historical versions of source code
- Allow multiple people or teams to work on it at the same time
- In case of any issue we can always go back to any of the previous revisions

Software Version Control

- Prevents developers from accidentally losing code due to laptop crashes
- Allows periodic checking in of code to hierarchical tree structure of folders with code in them
- Keeps tracks of who changed what and when via a process of tagging
- Allows concurrent checkins by multiple developers
- Allows multiple versions of code to be maintained via the process of branching
- This is useful when different features to be delivered to different customers
- If both features are part of main project or product, the sub branches can be merged into the main or master branch

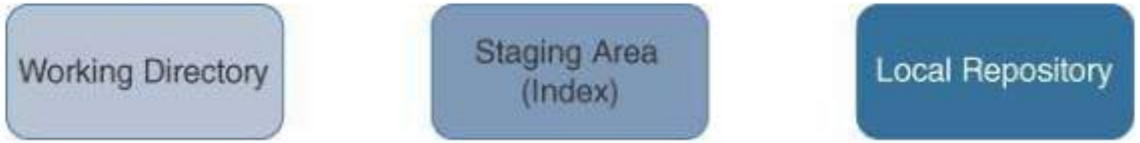
IP ROUTE TECHNOLOGIES PVT LTD

Git

- Git is the most popular and widely used Software Version Control System
- It is free and open source software
- Created by Linus Torvalds who is also the creator of Linux
- Git is a distributed version control system known for its speed and scalability

Git (Cont.)

- Git uses the traditional file system like structure to track the changes to files
- Keeps track of the following main structures or trees
    - Working directory
        - Local directory where all the code, binaries, images, docs etc. are stored
    - Staging area
        - Internal storage area for items to be synced (new and changed)
    - Local reposit
        - Internal sto



Git (Cont.)

- Every file being managed by Git has a status attached to it
- Its goes through a status life cycle based on the modifications happening
- The status of the file at any point of time determines how Git handles the file

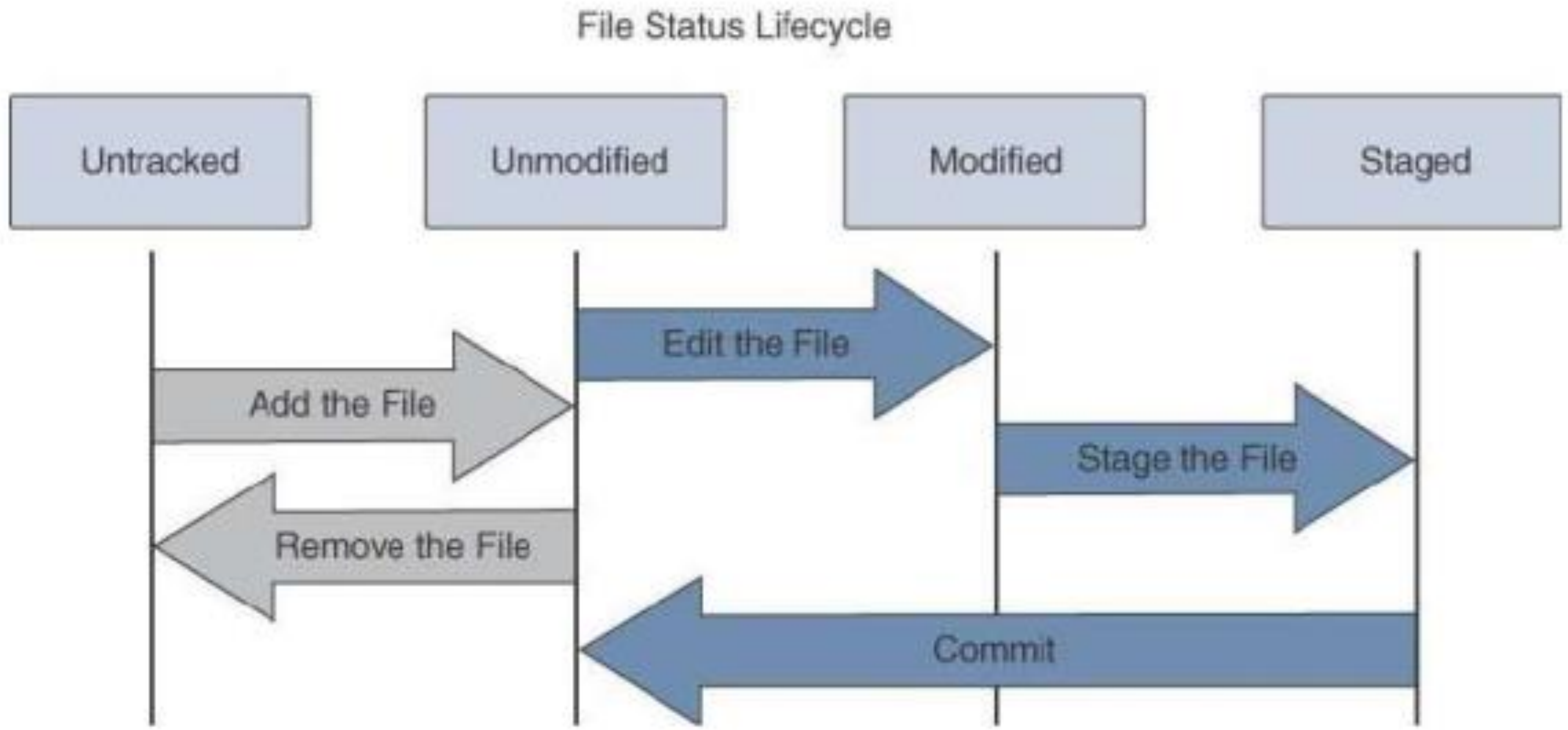
Git - File Status Life Cycle

- Untracked
  - Any file that is created in a dir that is managed by git is in this status
  - Git sees untracked files but do not do version control on them
  - For them to be tracked, we have to explicitly tell git to do so using ***git add <filename>*** command
- Unmodified
  - Git is watching these files for changes but did not find any

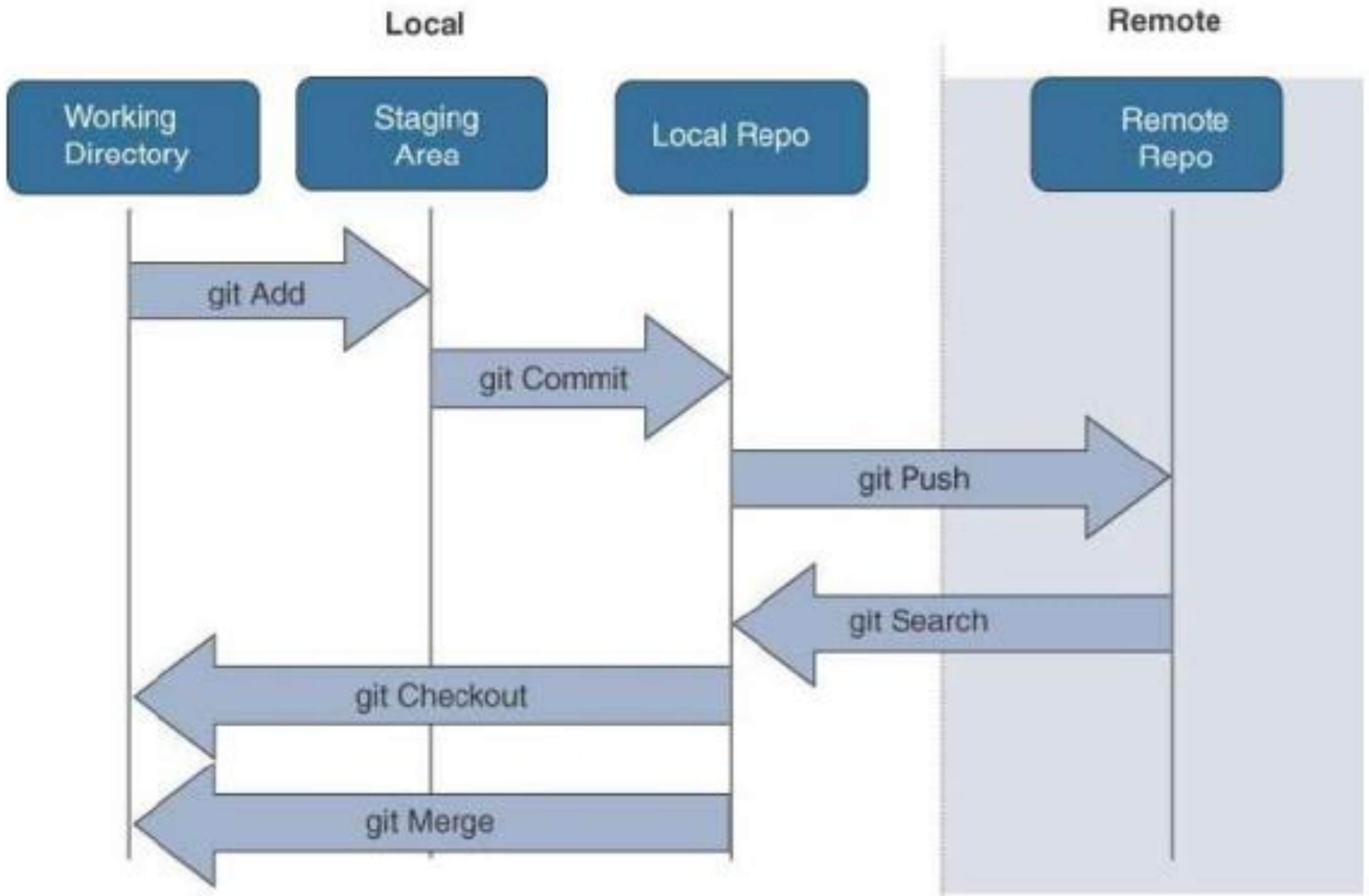
Git - File Status Life Cycle

- Modified
  - Any file tracked by Git which has undergone a modification
  - Modified files are currently being worked upon
  - Once modifications are done, we need to use ***git add***
- Staged
  - Modified file which is added to the index or staging area
  - Ready to be committed (pushed to local repository)
  - We need to commit using ***git commit***

Git - File Status Life Cycle



Git Workflow



IP ROUTE TECHNOLOGIES PVT LTD

Working with Git

- In Linux we can install git using apt or yum or any package manager based on the Linux flavor
- In windows we can download and install git from https://gitforwindows.org/
- Once installed we can start using git from the CLI or UI

Git Configuration

- Before we can start using Git to commit the code, we need to configure username and email
- The username and email are just labels and are used just to track the commits
- They have nothing to do with the email used to register hosted services like github or gitlab or anything else
- However git makes it mandatory for these details to be provided
- Configuration can be global (across repos) or repo specific

Git Configuration

- Global config is shown below

```
$ git config --global user.name DevNetAdmin
$ git config --global user.email devnet_admin@octa.com
$ git config --list
user.email=devnet_admin@octa.com
user.name=DevNetAdmin
http.sslcainfo=/etc/ssl/certs/ca-certificates.crt
```

- Local or repo specific config is shown below

```
$ git config user.name DevNetUser
$ git config user.email devnet_user@octa.com
```

Tracking an existing folder

- cd to an existing project dir and use the command ***git status***
- Since git does not know about this folder, you should see an error like below

```
$ pwd
/home/nexadmin/work/ccna_devnet/git_test
$ git status
fatal: not a git repository (or any of the parent directories): .git
$
```

Tracking an existing folder

- Use the command ***git init*** to tell git to start tracking this folder
- Use the command ***git status*** again

```
$ git init
Initialized empty Git repository in /home/nexadmin/work/ccna_devnet/git_test/.git/
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
$
```

Tracking an existing folder

- ***git init*** command made the current directory as a git directory
- By default it creates a branch named ***master***
- ***git status*** show that there is nothing changed hence nothing to do
- Copy or create a new file in this directory and use ***git status*** again

```
$ touch new_file.txt
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        new_file.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Tracking an existing folder

- New file is in ***untracked*** status
- Follow the instruction given and do git add <filename>
- This moves the file to staging area, ready to be committed

```
$ git add new_file.txt
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   new_file.txt
```

Tracking an existing folder

- If we added by mistake we can remove it using the command shown
- Else we can commit it using the command git commit -m <comments>

```
$ git commit -m "new file added"
[master (root-commit) 377d9ea] new file added
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 new_file.txt
$ git status
On branch master
nothing to commit, working tree clean
$
```

Tracking an existing folder

- Use ***git log*** to check what who checked in what and when
- We have successfully version controlled a blank file :)
- From now on the process is same for every change/ edit that happens to the file

```
$ git log
commit 377d9ea7f393ac54c2dda2a18976712e8a46b16d (HEAD -> master)
Author: Nagaraj <nravinuthala@gmail.com>
Date:   Wed Jan 4 06:14:11 2023 -0500

       new file added
$
```

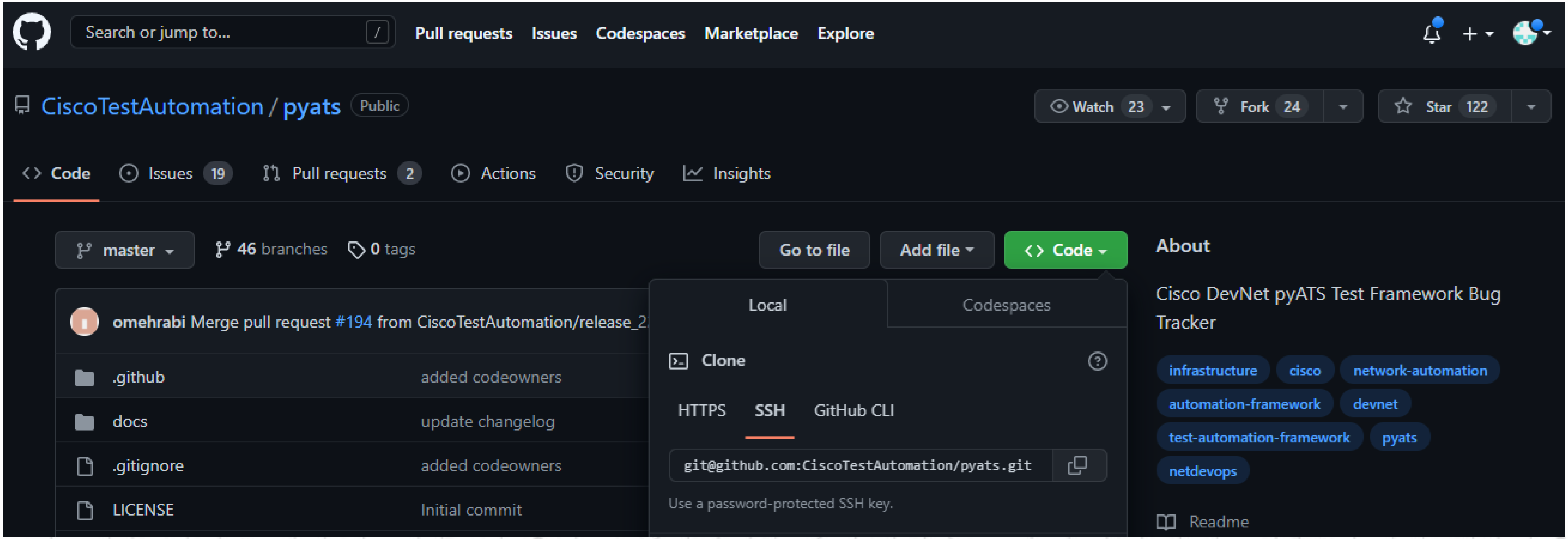
Remote Repositories

- What we did so far
  - Created a local git repo with some files under it
- This still runs the risk of losing the content if the system crashes
- The solution is in git itself as it is a distributed version control system
- Meaning there is a remote repository corresponding to the local repository
- Github or Gitlab are hosted applications based on git and support remote repositories

Cloning a repository

- We can clone a remote repository from github to loal using the command git clone
- But first we need the repository URL
- Go to github, login and search for the repository you are interested in
- Once found, look for a green button named Code and click on it
- You will see 3 options, HTTPS, SSH and GitHub CLI
- These are 3 ways of interacting with remote repositories
- SSH is the preferred means of interaction
- For SSH connectivity, we need to generate an SSH key on the client and add that key to github

Cloning a repository



omehrabi Merge pull request #194 from CiscoTestAutomation/release_2

.github	added codeowners
docs	update changelog
.gitignore	added codeowners
LICENSE	Initial commit

```
git@github.com:CiscoTestAutomation/pyats.git
```

Use a password-protected SSH key.

Cisco DevNet pyATS Test Framework Bug Tracker

SSH Key Management

- Generating SSH Key
  - https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent
- Adding ssh key to github
  - https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account

Cloning a repository

- Shown below is the output of cloning the repository using SSH connection

```
$ git clone git@github.com:CiscoTestAutomation/pyats.git
Cloning into 'pyats'...
remote: Enumerating objects: 1292, done.
remote: Counting objects: 100% (215/215), done.
remote: Compressing objects: 100% (168/168), done.
remote: Total 1292 (delta 97), reused 78 (delta 46), pack-reused 1077
Receiving objects: 100% (1292/1292), 2.01 MiB | 878.00 KiB/s, done.
Resolving deltas: 100% (717/717), done.
$
```

Cloning a repository

- Once a repo is cloned, it sits in your local file system like a local project being managed by git as a local repository
- We can work on any of the files using your favourite IDEs
- Once any existing files are changed/ removed or new files are added, committing them to local repo is same as steps mentioned above

Pushing and Pulling Files

- Pushing files is the process of syncing new files or changes from local repo to remote repo
- Pulling is the reverse process of pushing - refers to getting latest changes from remote to local
- To be able to do this, first the remote repo details should be configured with the local repo
- When we clone a repo, these details are automatically added
- Verify it using the command ***git remote -v***

Pushing and Pulling Files

- If remote repo is already configured, we should see something like below
- If not we can configure it as follows

```
$ git remote -v
origin  git@github.com:CiscoTestAutomation/pyats.git (fetch)
origin  git@github.com:CiscoTestAutomation/pyats.git (push)
$
```

```
$ git remote add remote_repo git@github.com:CiscoTestAutomation/pyats.git
$ git remote -v
origin  git@github.com:CiscoTestAutomation/pyats.git (fetch)
origin  git@github.com:CiscoTestAutomation/pyats.git (push)
remote_repo    git@github.com:CiscoTestAutomation/pyats.git (fetch)
remote_repo    git@github.com:CiscoTestAutomation/pyats.git (push)
$
```

Pushing and Pulling Files

- If remote repo is already configured, we should see something like below
- If not we can configure it as follows

```
$ git remote -v
origin  git@github.com:CiscoTestAutomation/pyats.git (fetch)
origin  git@github.com:CiscoTestAutomation/pyats.git (push)
$
```

```
$ git remote add remote_repo git@github.com:CiscoTestAutomation/pyats.git
$ git remote -v
origin  git@github.com:CiscoTestAutomation/pyats.git (fetch)
origin  git@github.com:CiscoTestAutomation/pyats.git (push)
remote_repo    git@github.com:CiscoTestAutomation/pyats.git (fetch)
remote_repo    git@github.com:CiscoTestAutomation/pyats.git (push)
$
```

Reverting to a previous commit

- We know that git is all about revisions of code in the repo
- So it should be possible to go back to a previous revision if needed
- The commands ***git reset*** and ***git revert*** will help us do this
- Git Reset will not preserve commit history and overwrites files and hence there is a risk of losing someone else’s changes
- So reset is typically used in local repo to revert individual user changes
- In a distributed env git revert is preferred as it preserves commit history

Reverting to a previous commit

- Create a sample file and make 2 or 3 commits

```
$ git log --oneline
b38670c (HEAD -> master) commit3
d37f1a7 commit2
f776383 commit1
$ cat test_file.txt
line 1
line 2
line 3
```

Reverting to a previous commit

- Revert the last commit and check the file contents and git log
- Note that commit history is preserved

```
$ git revert b38670c
[master 2bc3f70] Revert "commit3"
 1 file changed, 1 insertion(+), 1 deletion(-)
nexadmin@DESKTOP-89IJ1T7:~/work/ccna_devnet/git_test2$ cat test_file.txt
line 1
line 2

$ git log --oneline
2bc3f70 (HEAD -> master) Revert "commit3"
b38670c commit3
d37f1a7 commit2
f776383 commit1
```

Reverting to a previous commit

- Now do a git reset and give commit id of the first commit with an option –hard
- File is overwritten with commit 1 version and commit history is lost as well
- Hence git reset should be used with caution and should ideally be limited to local repository

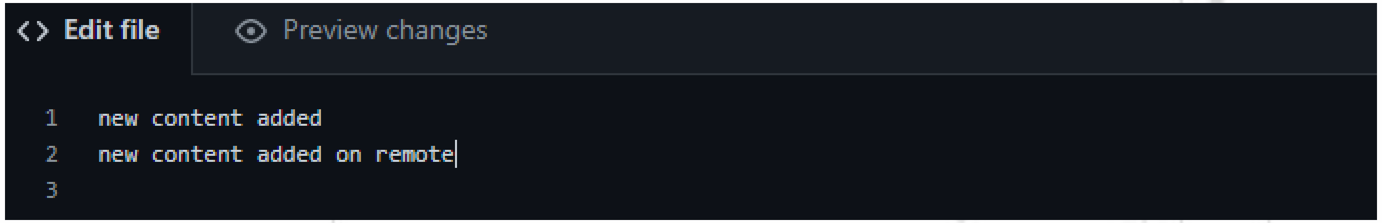
```
$ git reset f776383 --hard
HEAD is now at f776383 commit1
$ cat test_file.txt
line 1
$ git log --oneline
f776383 (HEAD -> master) commit1
```

Syncing changes from remote repo

- We have already seen ***git pull*** which syncs local with remote
- This gets the changes from the remote repo and updates the local copy of the remote repo as well as the local repo
- In some cases, you may want to just get the latest changes from remote repo but not update the local repo
- In such cases we can use ***git fetch***
- This gets the changes from remote and updated the local copy of the remote repo but not the local repo
- After this we have to do a ***git merge*** to update the local repo
- ***git fetch*** + ***git merge*** is considered to be safer then git pull

Syncing changes from remote repo

```
$ cat new_file.txt
new content added
```



```
new content added
new content added on remote
```

```
$ git fetch git_test_remote master
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 664 bytes | 664.00 KiB/s, done.
From github.com:nravinuthala/git_test
 * branch              master  -> FETCH_HEAD
   a16c203..c335731  master   -> git_test_remote/master
```

Syncing changes from remote repo

```
$ cat new_file.txt
new content added
```

```
$ git merge git_test_remote/master  master
Updating a16c203..c335731
Fast-forward
 new_file.txt | 1 +
 1 file changed, 1 insertion(+)
$ cat new_file.txt
new content added
new content added on remote
```

Working with Branches

- Branching is another useful feature of Git
- Support there is a defect in the product given to a customer, we need to fix the defect
- At the same time a new feature development is happening
- So we create 2 branches, one for defect fix and other for new feature development
- This is to ensure that the new feature development will not cause more issues for the customer who already has a previous version working well
- Branches can be created in either of the following ways
  - ○ git checkout -b <branch name>
  - ○ git branch <branch name>

Working with Branches

- Using git branch, new branch is created but we are still in old branch
- Need to checkout to change to new branch

```
$ git branch
* master
$ git branch bugfix
$ git branch
  bugfix
* master
```

```
$ git checkout bugfix
Switched to branch 'bugfix'
$ git branch
* bugfix
  master
```

Working with Branches

- The command git checkout -b does this in a single step

```
$ git checkout -b new_feature
Switched to a new branch 'new_feature'
$ git branch
  bugfix
  master
* new_feature
```

Working with Branches

- Once the bug or new feature is tested and is working fine, we can merge those branches with master and delete them

```
$ cat new_file.txt
new content added
new content added on remote
$ git merge bugfix
Updating c335731..8e20ed6
Fast-forward
 new_file.txt | 1 +
 1 file changed, 1 insertion(+)
$ cat new_file.txt
new content added
new content added on remote
defect fixed
```

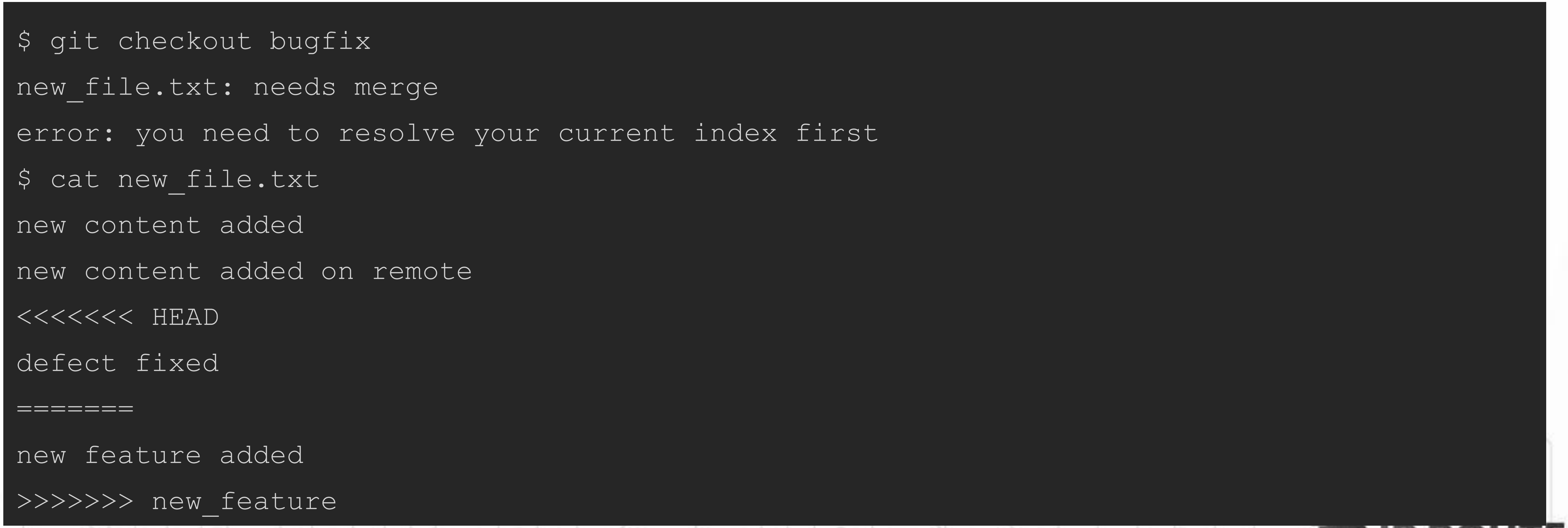
Working with Branches

- If same file is modified by two people or in two branches we will have a conflict

```
$ git checkout new_feature
new_file.txt: needs merge
error: you need to resolve your current index first
$ cat new_file.txt
new content added
new content added on remote
<<<<<<< HEAD
defect fixed
=======
new feature added
>>>>>>> new_feature
```

Working with Branches

- If same file is modified by two people or in two branches we will have a conflict



```
$ git checkout bugfix
new_file.txt: needs merge
error: you need to resolve your current index first
$ cat new_file.txt
new content added
new content added on remote
<<<<<<< HEAD
defect fixed
=======
new feature added
>>>>>>> new_feature
```

Working with Branches

- Git adds some lines to highlight the conflicting parts
- We can decide what to do, delete parts added by git and commit the file

```
new content added
new content added on remote
<<<<<<< HEAD
defect fixed
=======
new feature added
>>>>>>> new_feature
```

```
new content added
new content added on remote
defect fixed
new feature added
```

Working with Branches

- We decided to keep both bug fix and new feature
- Save and commit changes
- Branches can now be deleted

```
$ git add .
$ git commit -m "resolved conflict"
[master e223b6f] resolved conflict
$ git branch -d bugfix
Deleted branch bugfix (was 8e20ed6).
$ git branch -d new_feature
Deleted branch new_feature (was 54f242c).
```

Comparing commits with diff

- git diff show difference between local repo and staging
- Make some change to the file in working dir and do not stage it and do a diff

```
$ git diff
diff --git a/new_file.txt b/new_file.txt
index 27bf150..477307c 100644
--- a/new_file.txt
+++ b/new_file.txt
@@ -2,3 +2,4 @@ new content added
 new content added on remote
 defect fixed
 new feature added
+new code added
```

Comparing commits with diff

- git diff –cached show difference between staging and last commit
- Stage the file using git add and do a diff

```
$ git diff --cached
diff --git a/new_file.txt b/new_file.txt
index 27bf150..477307c 100644
--- a/new_file.txt
+++ b/new_file.txt
@@ -2,3 +2,4 @@ new content added
 new content added on remote
 defect fixed
 new feature added
+new code added
```

Comparing commits with diff

- git diff HEAD show difference between working directory and last commit
- Useful to know effect of next commit on the local repo

```
$ git diff HEAD
diff --git a/new_file.txt b/new_file.txt
index 27bf150..477307c 100644
--- a/new_file.txt
+++ b/new_file.txt
@@ -2,3 +2,4 @@ new content added
 new content added on remote
 defect fixed
 new feature added
+new code added
```

Comparing commits with diff

- Diff can also be used to check difference between current and target branch
- git diff <branch_name> <file_name> shows difference in file_name between current and branch_name

```
$ git diff bugfix new_file.txt
```