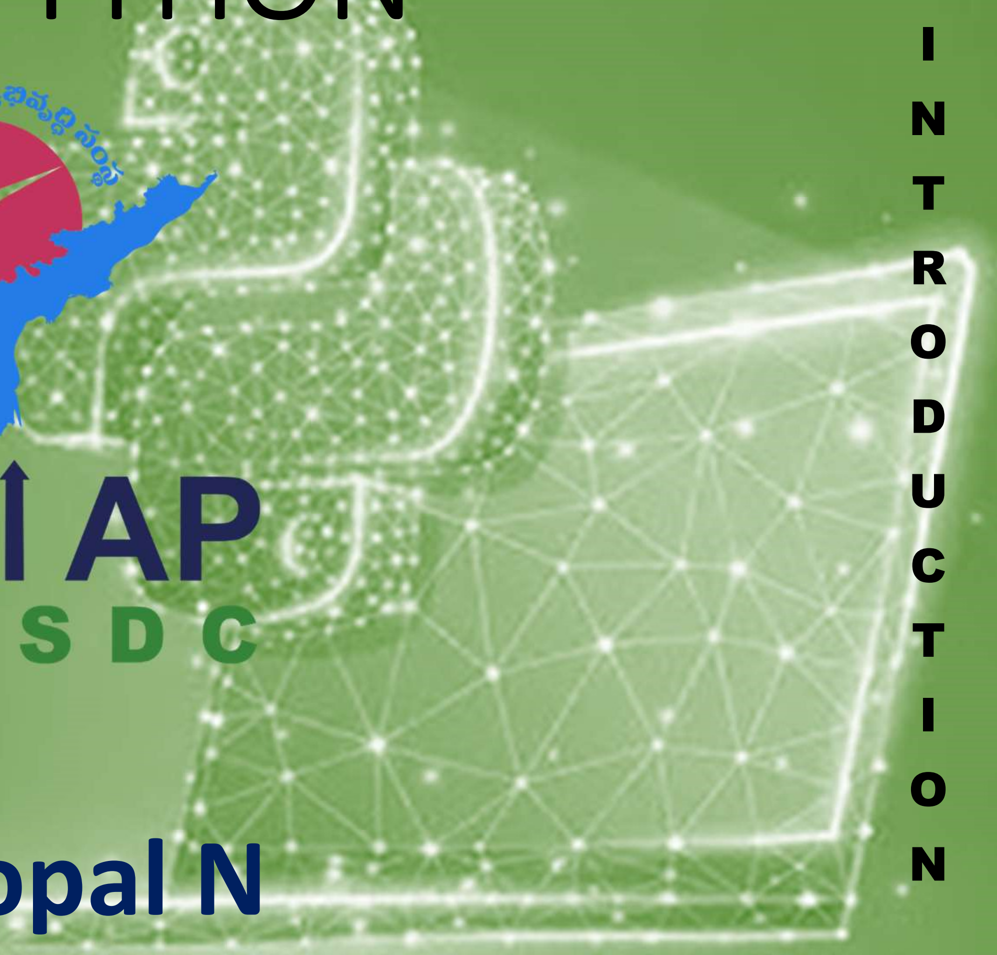
BASICS OF PYTHON

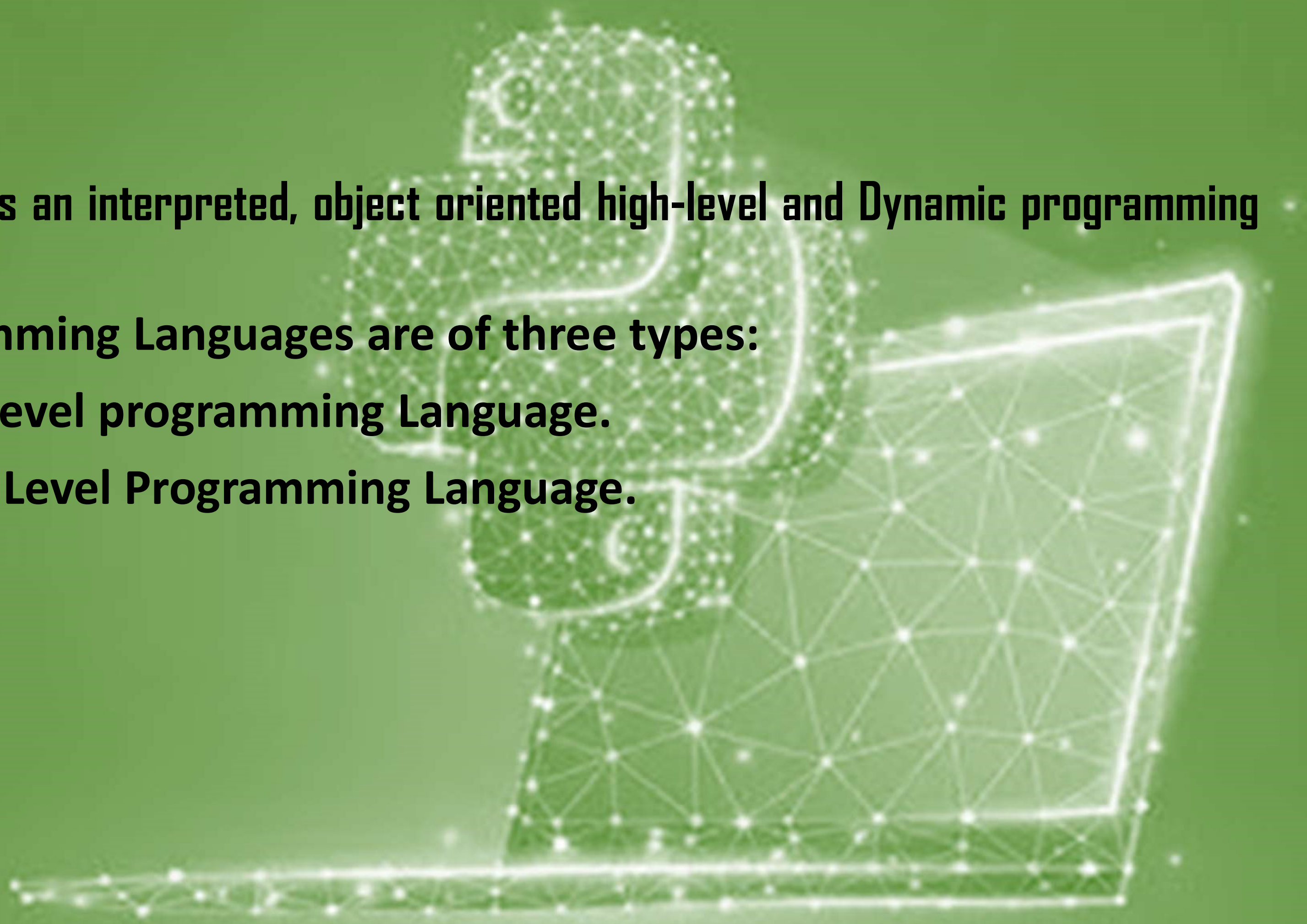
INTRODUCTION



By Venugopal N

INTRODUCTION

PYTHON

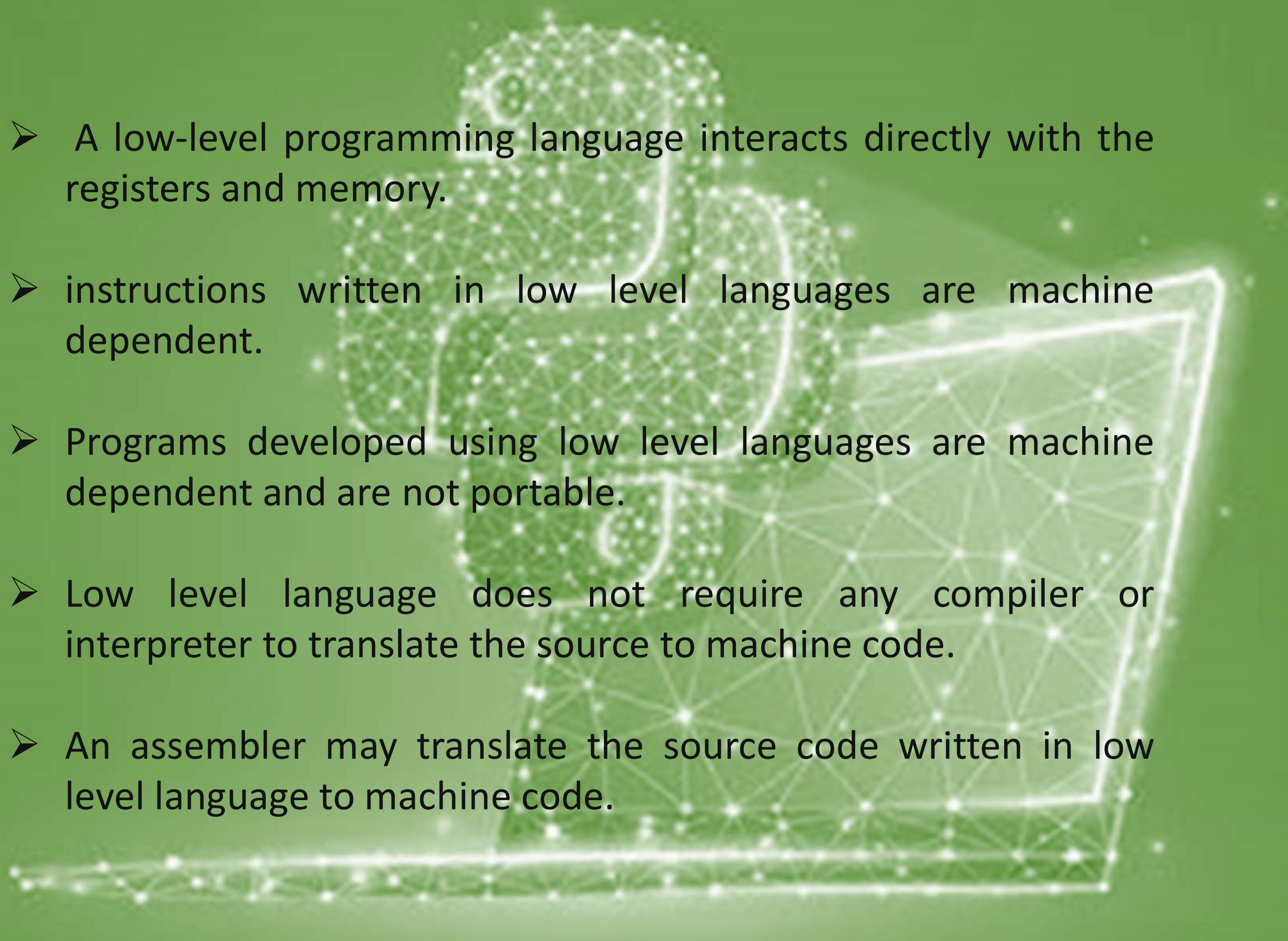


- PYTHON is an interpreted, object oriented high-level and Dynamic programming language
- **Programming Languages are of three types:**

  1. Low-level programming Language.

  2. High- Level Programming Language.

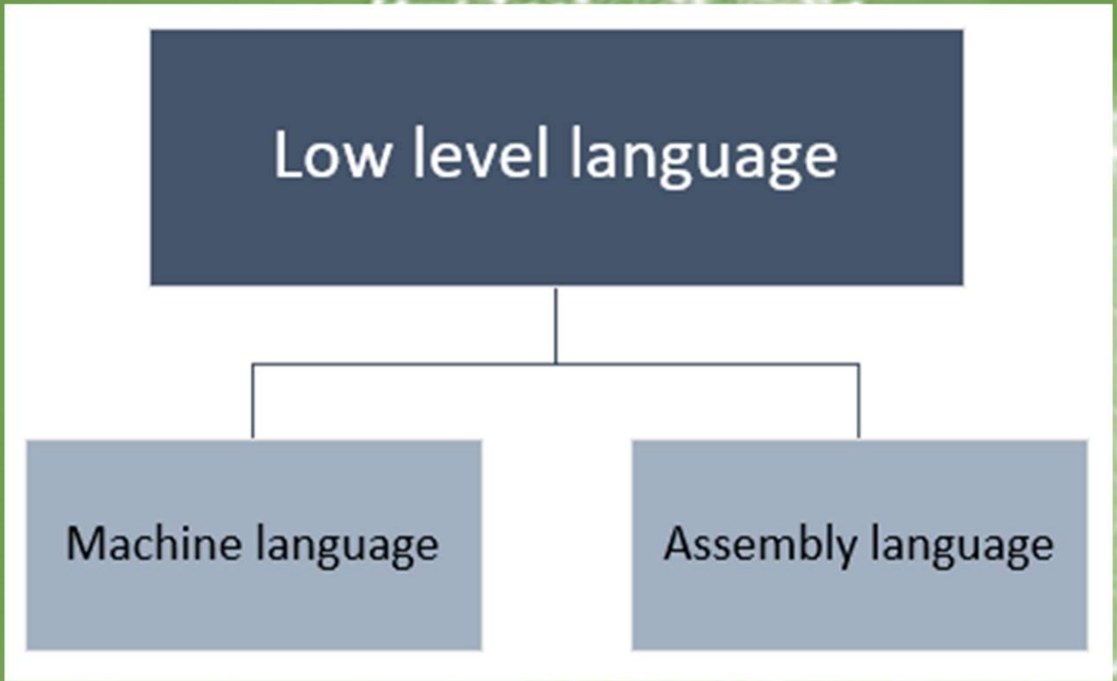
WHAT IS LOW LEVEL LANGUAGE ?



- A low-level programming language interacts directly with the registers and memory.
- instructions written in low level languages are machine dependent.
- Programs developed using low level languages are machine dependent and are not portable.
- Low level language does not require any compiler or interpreter to translate the source to machine code.
- An assembler may translate the source code written in low level language to machine code.

TYPES OF

LOW LEVEL LANGUAGES



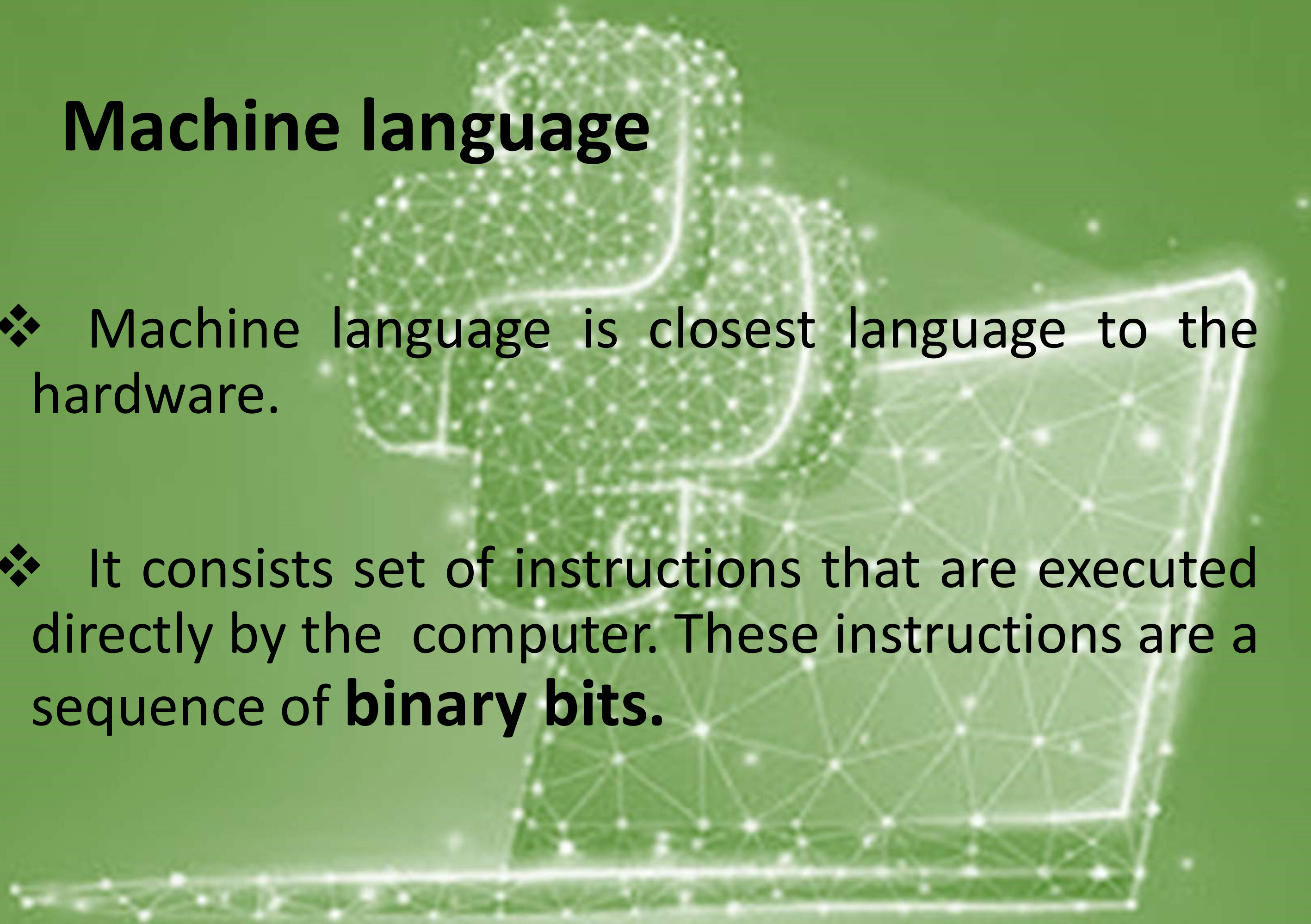
Low level language

Machine language

Assembly language

WHAT IS
M
A
C
H
I
N
E
LANGUAGE

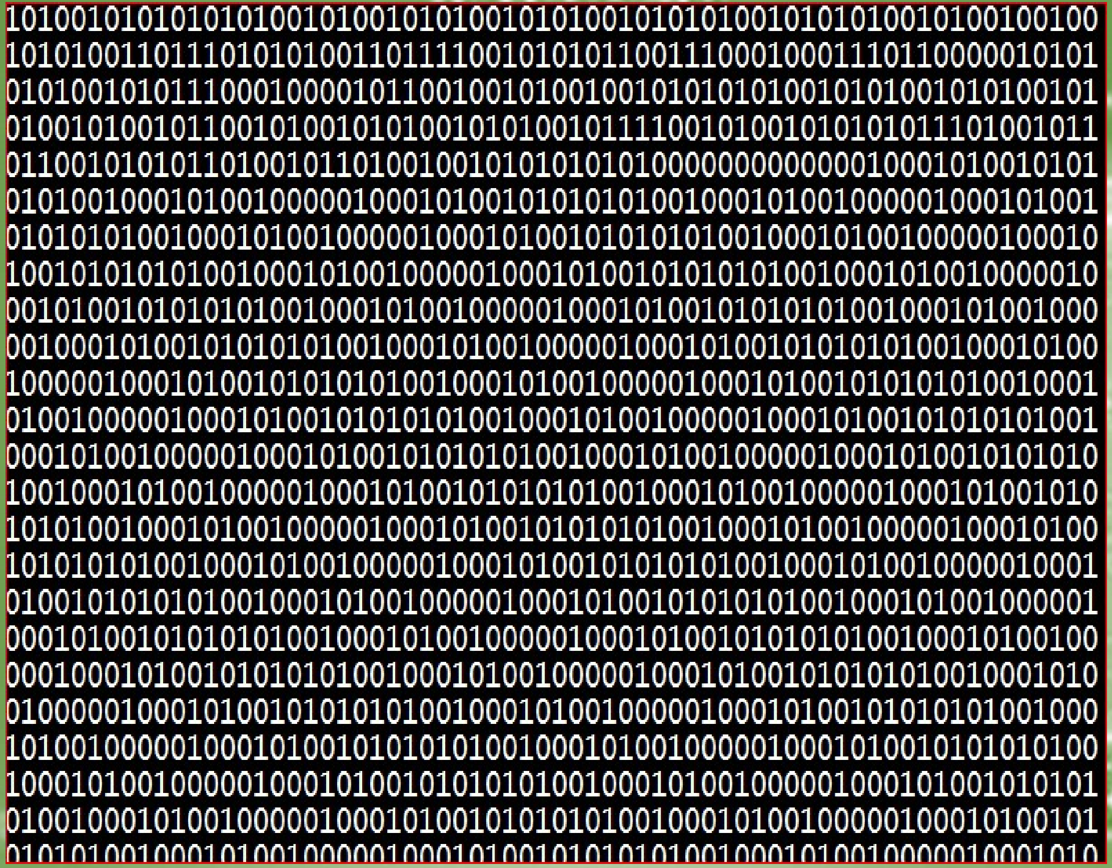
Machine language



- ❖ Machine language is closest language to the hardware.

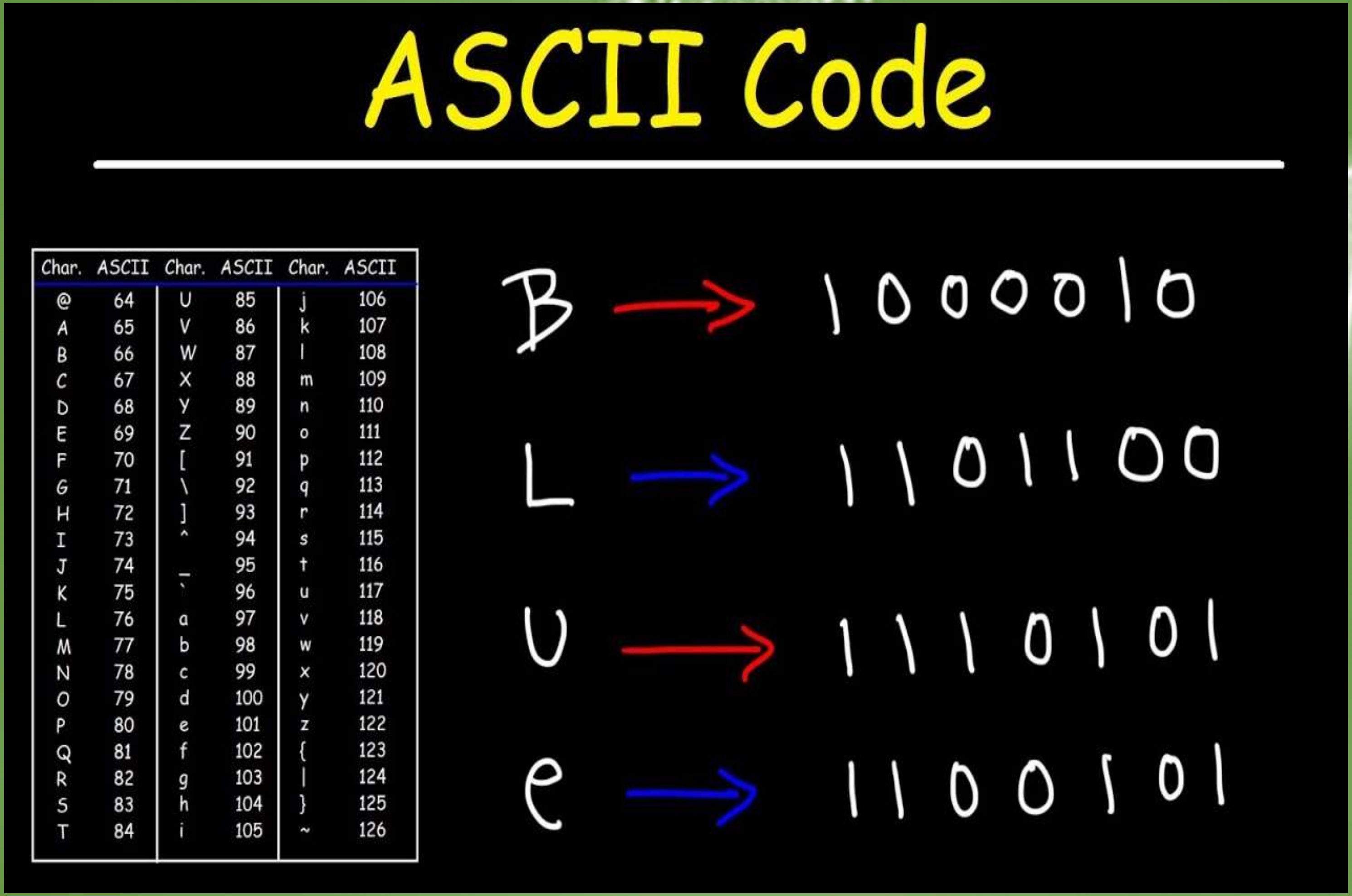
- ❖ It consists set of instructions that are executed directly by the computer. These instructions are a sequence of **binary bits.**

EXAMPLE OF M A C H I N E LANGUAGE



```
10100101010101010100101001010100101010010101010010101010010100100100
1010100110111010101001101111001010101100111000100011101100000010101
010100101011100010000101100100101001001010101010010101001010100101
01001010010110010100101010010101001011110010100101010101110100 1011
0110010101011010010110100100101010101010000000000000010001010010101
010100100010100100000100010100101010101001000101001000001000101001
010101010010001010010000010001010010101010100100010100100000100010
100101010101001000101001000001000101001010101010010001010010000010
00101001010101010100100010100100000100010100101010101001000101001000
001000101001010101010010001010010000010001010010101010100100010100
100000100010100101010101001000101001000001000101001010101010010001
010010000010001010010101010100100010100100000100010100101010101001
000101001000001000101001010101010010001010010000010001010010101010
100100010100100000100010100101010101001000101001000001000101001010
101010010001010010000010001010010101010100100010100100000100010100
101010101001000101001000001000101001010101010010001010010000010001
010010101010100100010100100000100010100101010101001000101001000001
000101001010101010010001010010000010001010010101010100100010100100
000100010100101010101001000101001000001000101001010101010010001010
010000010001010010101010100100010100100000100010100101010101001000
101001000001000101001010101010010001010010000010001010010101010100
100010100100000100010100101010101001000101001000001000101001010101
010010001010010000010001010010101010100100010100100000100010100101
010101001000101001000001000101001010101010010001010010000010001010
```

ASCII Codes



ASCII Code

Char.	ASCII	Char.	ASCII	Char.	ASCII
@	64	U	85	j	106
A	65	V	86	k	107
B	66	W	87	l	108
C	67	X	88	m	109
D	68	Y	89	n	110
E	69	Z	90	o	111
F	70	[	91	p	112
G	71	\	92	q	113
H	72	]	93	r	114
I	73	^	94	s	115
J	74	_	95	t	116
K	75	`	96	u	117
L	76	a	97	v	118
M	77	b	98	w	119
N	78	c	99	x	120
O	79	d	100	y	121
P	80	e	101	z	122
Q	81	f	102	{	123
R	82	g	103	|	124
S	83	h	104	}	125
T	84	i	105	~	126

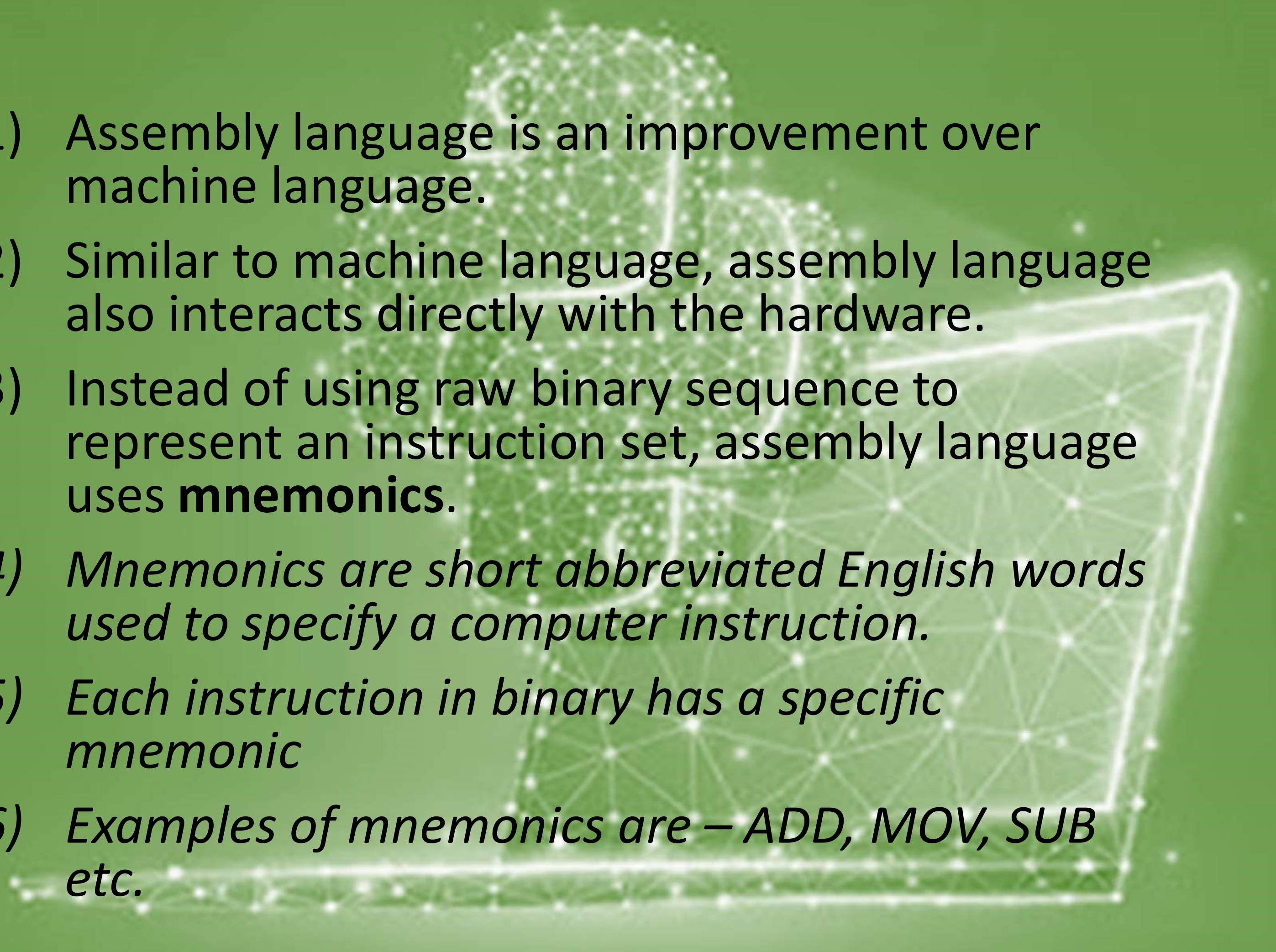
B → 1000010

L → 1101100

U → 1110101

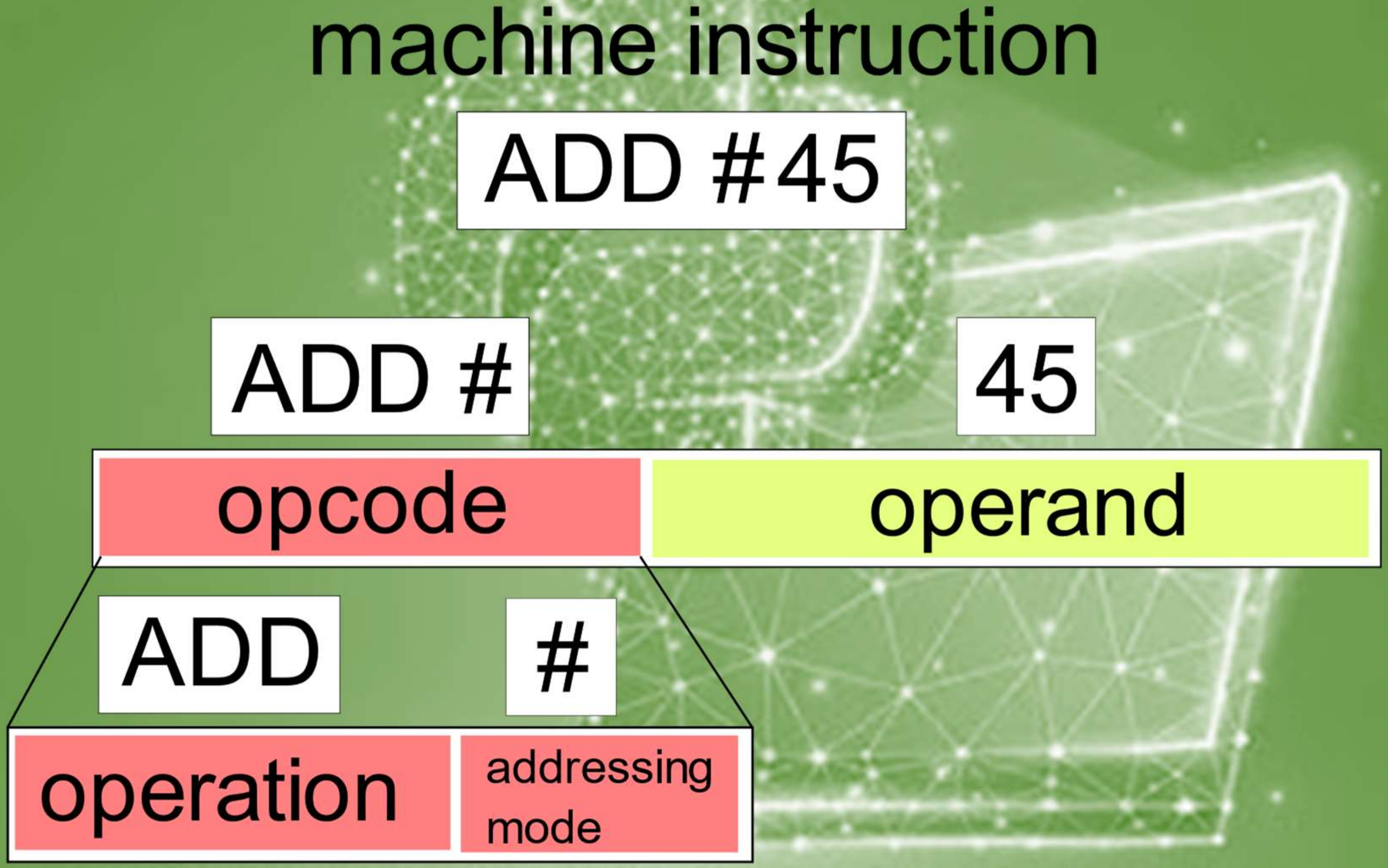
e → 1100101

WHAT IS ASSEMBLY LANGUAGE



1) Assembly language is an improvement over machine language.
2) Similar to machine language, assembly language also interacts directly with the hardware.
3) Instead of using raw binary sequence to represent an instruction set, assembly language uses **mnemonics**.
4) *Mnemonics are short abbreviated English words used to specify a computer instruction.*
5) *Each instruction in binary has a specific mnemonic*
6) *Examples of mnemonics are – ADD, MOV, SUB etc.*

Assembly Level Program Pattern



machine instruction

ADD #45

ADD #

45

opcode

operand

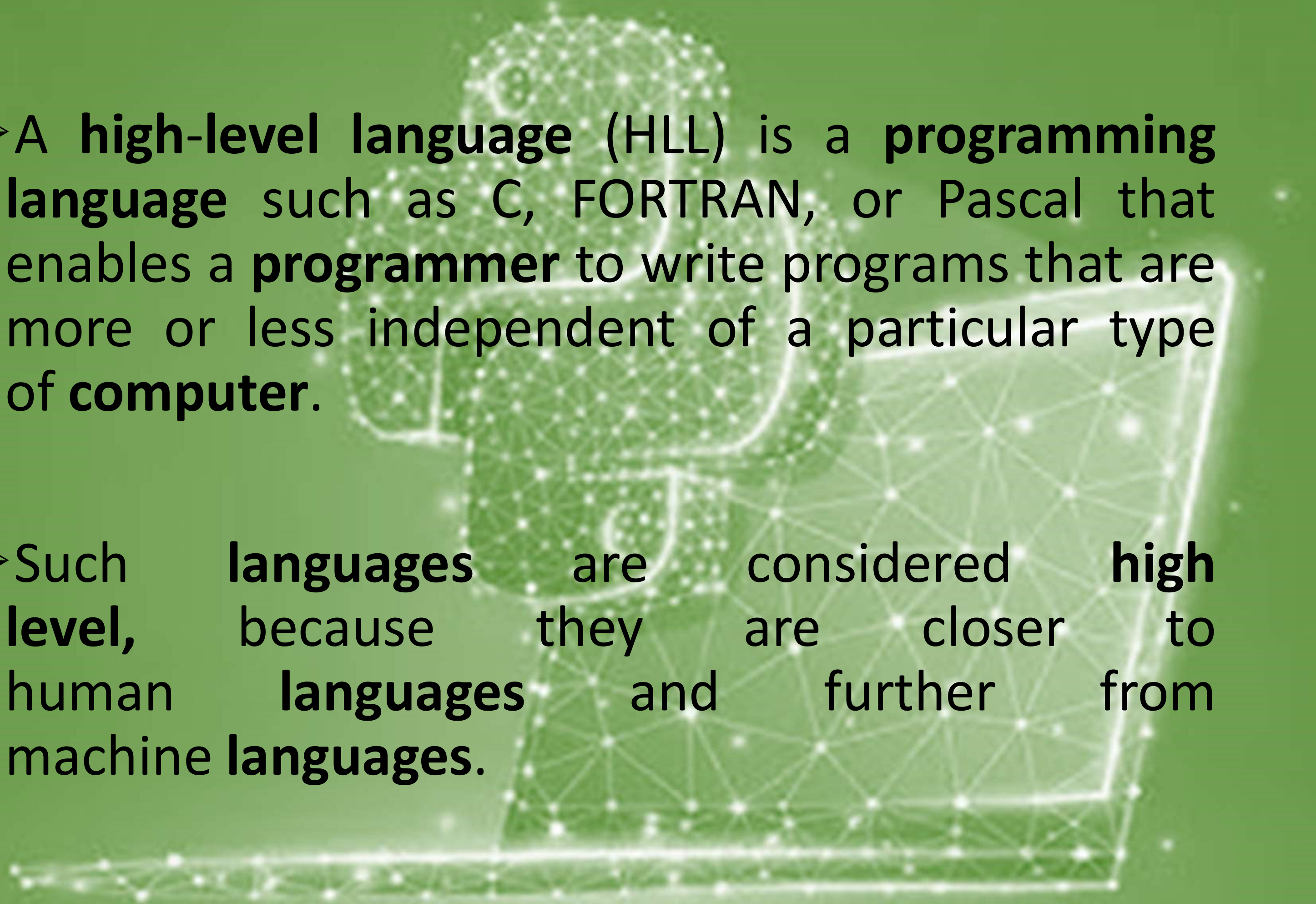
ADD

#

operation

addressing mode

WHAT IS HIGH LEVEL PROGRAMMING LANGUAGE



- A **high-level language** (HLL) is a **programming language** such as C, FORTRAN, or Pascal that enables a **programmer** to write programs that are more or less independent of a particular type of **computer**.

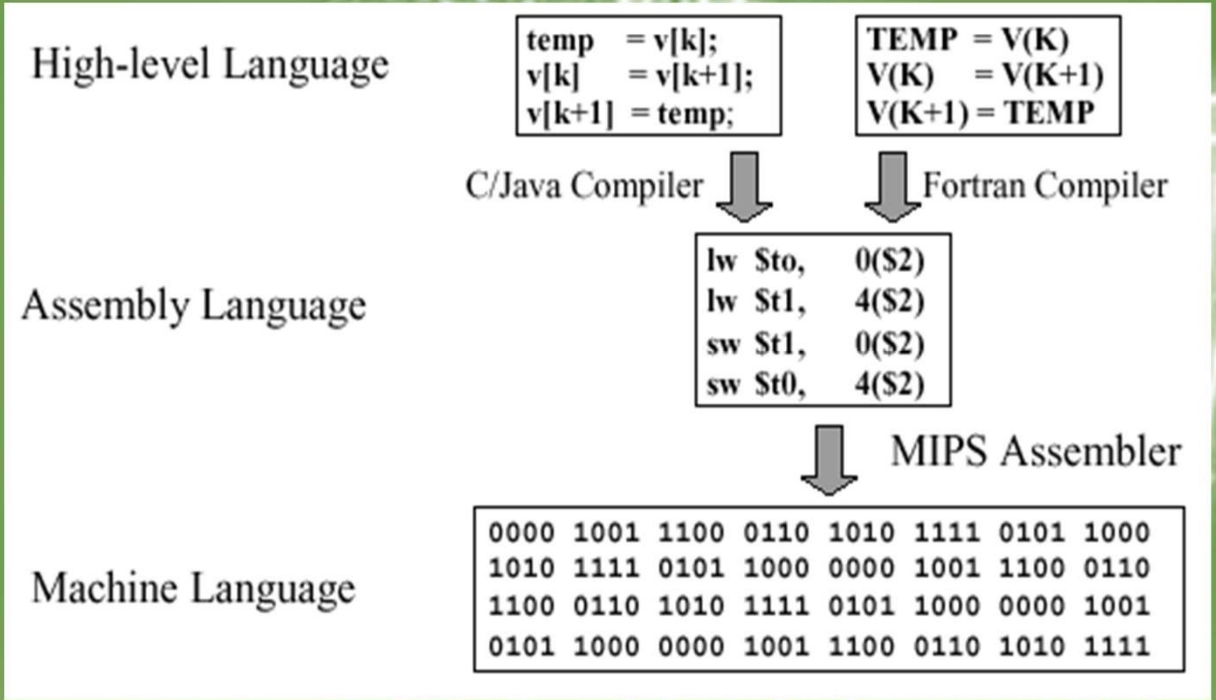
- Such **languages** are considered **high level,** because they are closer to human **languages** and further from machine **languages**.

PROGRAM

FLOW OF

HIGH
L
E
V
E
L
PROGRAMMING

LANGUAGE



High-level Language

```
temp    = v[k];
v[k]    = v[k+1];
v[k+1]  = temp;
```

```
TEMP  = V(K)
V(K)    = V(K+1)
V(K+1) = TEMP
```

C/Java Compiler

Fortran Compiler

Assembly Language

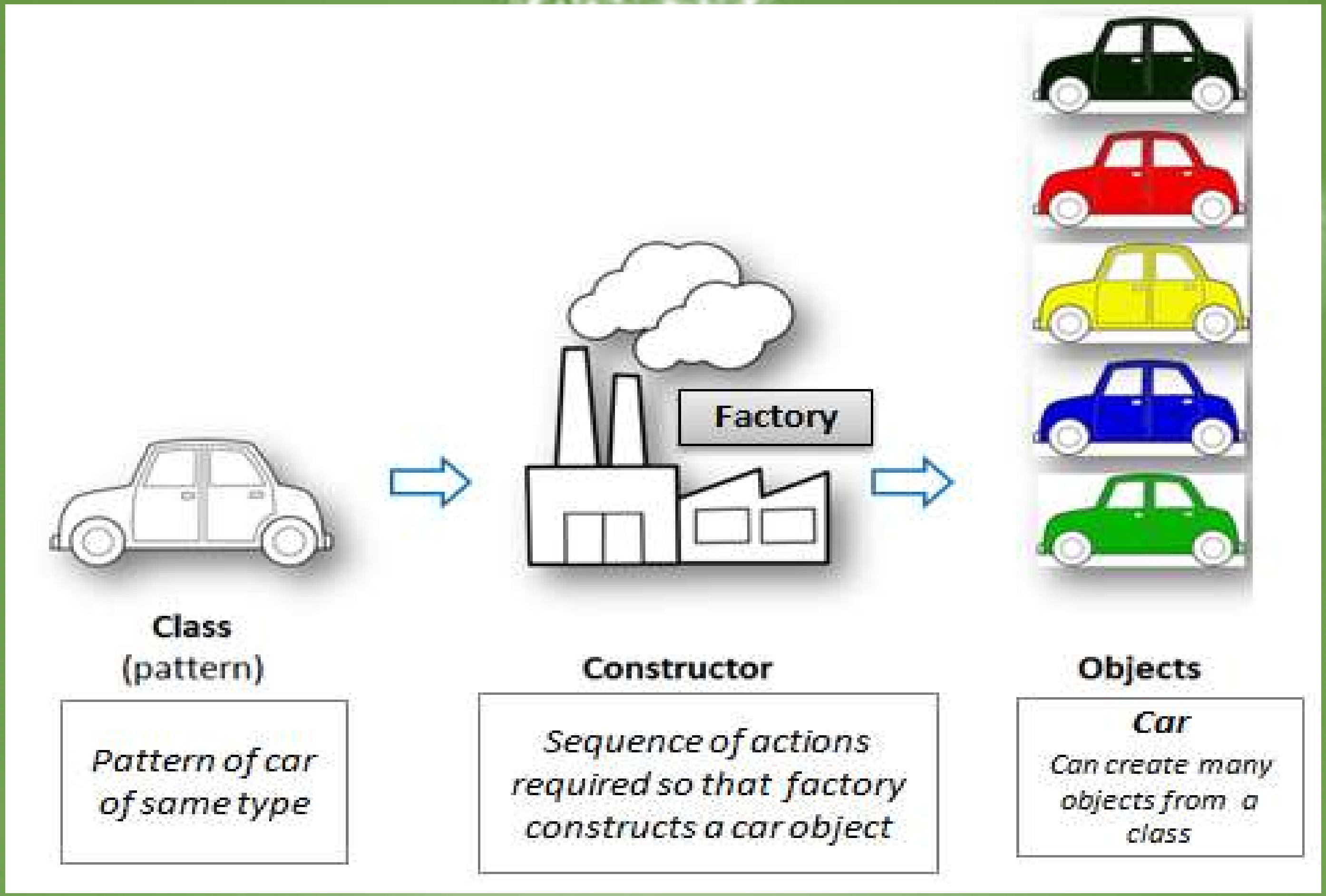
```
lw  $to,     0($2)
lw  $t1,     4($2)
sw  $t1,     0($2)
sw  $t0,     4($2)
```

MIPS Assembler

Machine Language

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

What is Meaning by Object Oriented



Class
(pattern)

Pattern of car of same type

Constructor

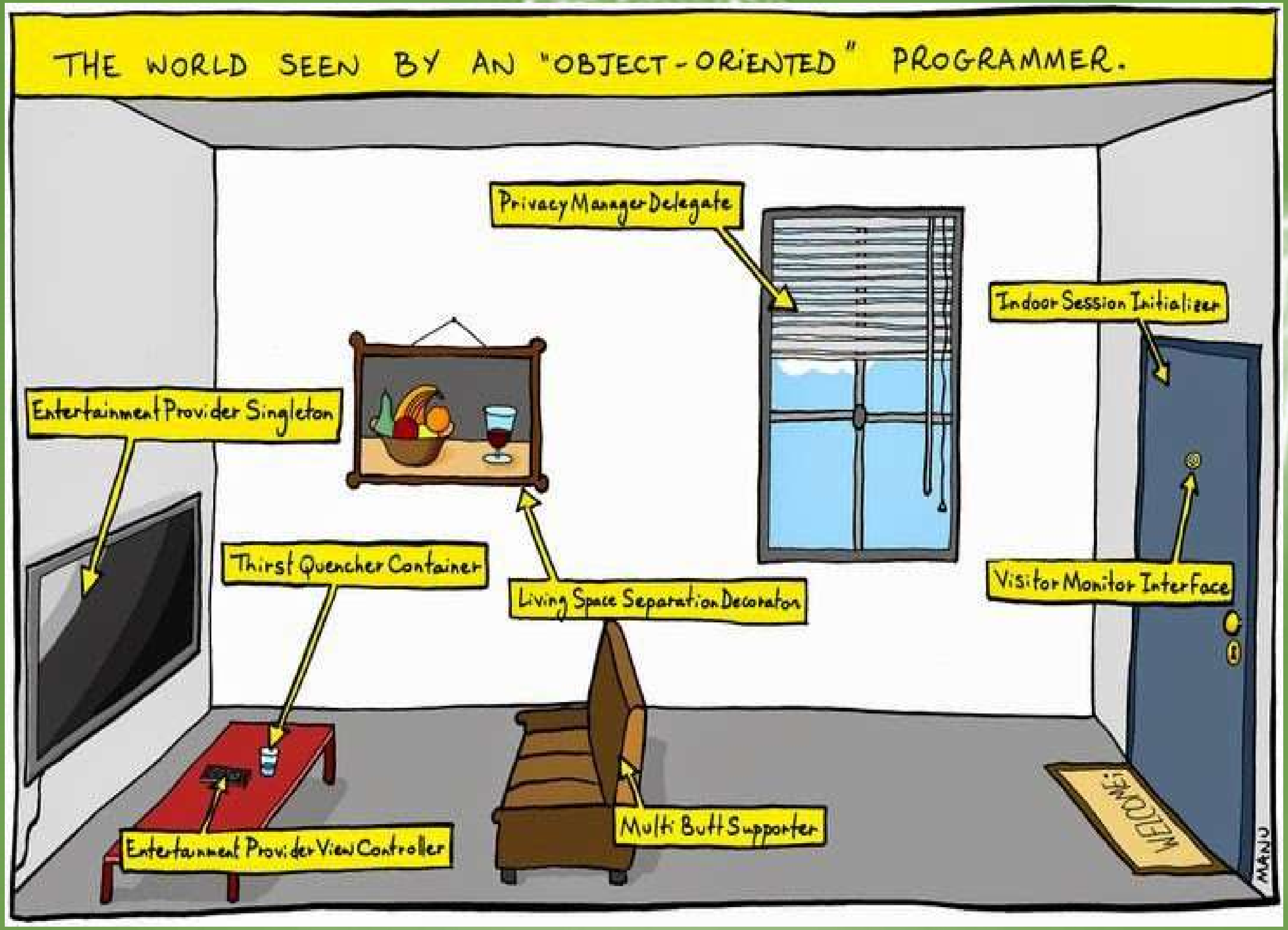
Sequence of actions required so that factory constructs a car object

Objects

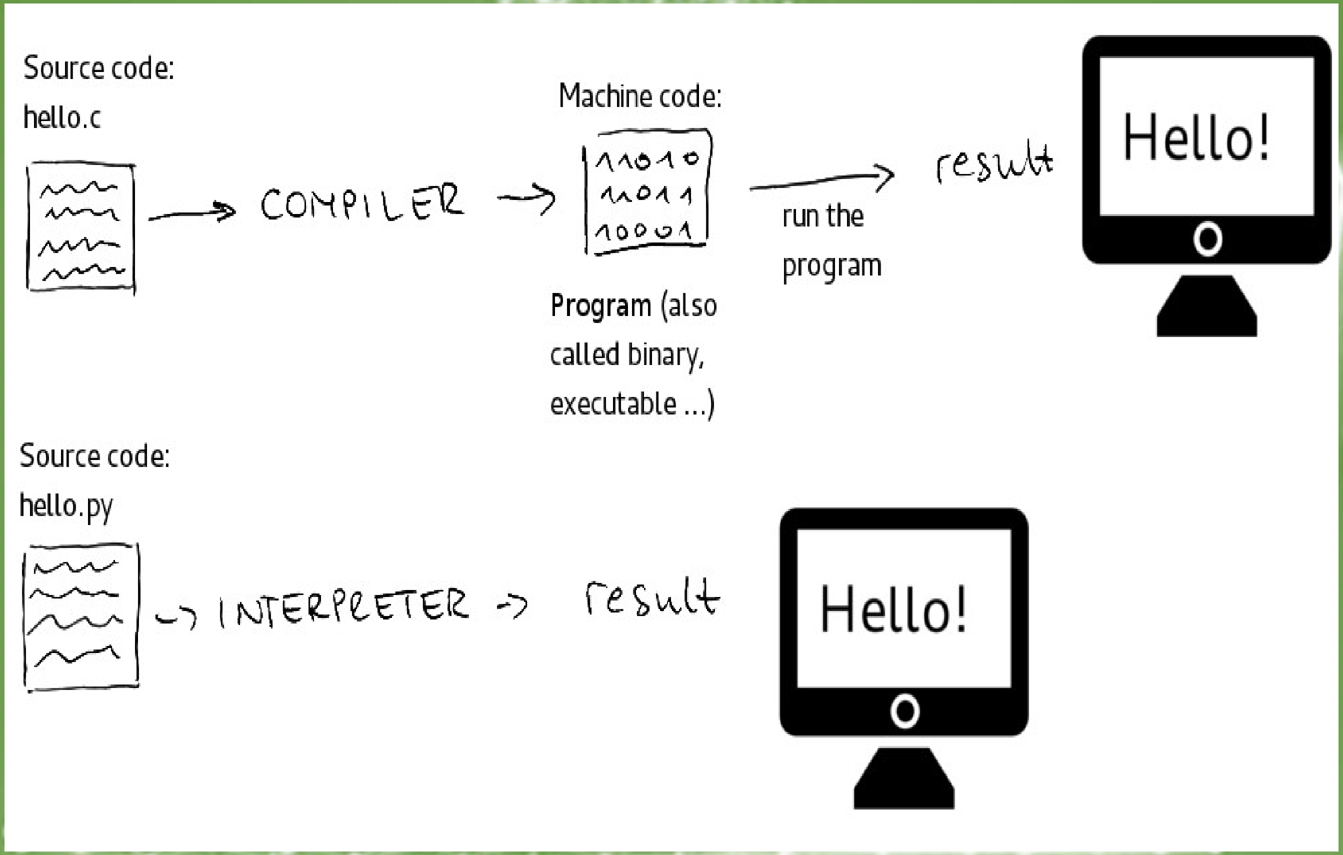
Car

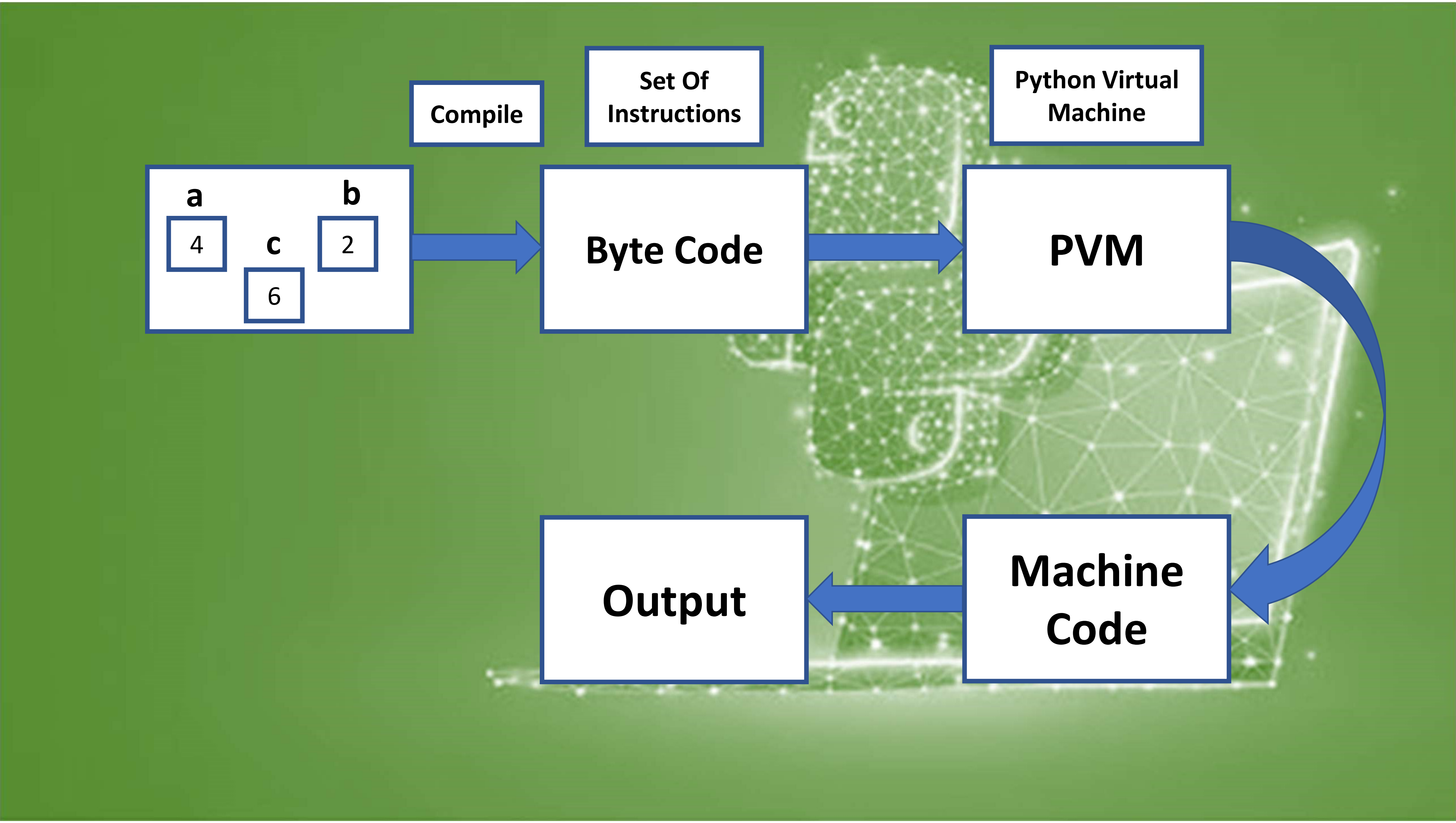
Can create many objects from a class

What is Meaning by Object Oriented

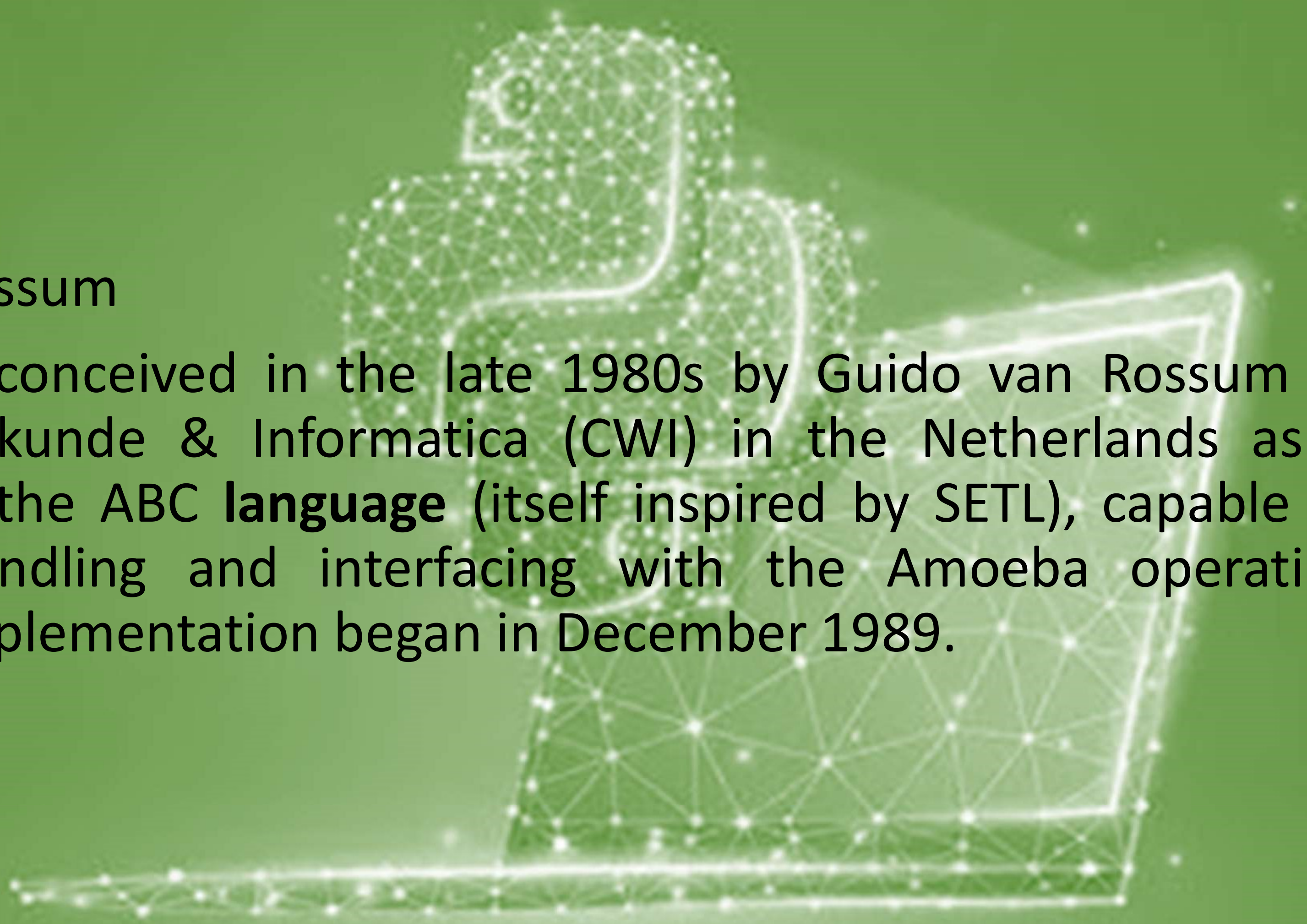

THE WORLD SEEN BY AN "OBJECT-ORIENTED" PROGRAMMER.
Privacy Manager Delegate
Indoor Session Initializer
Entertainment Provider Singleton
Thirst Quencher Container
Living Space Separation Decorator
Visitor Monitor Interface
Entertainment Provider View Controller
Multi Butt Supporter
WELCOME!
MANU

What is Meaning by Interpreted




Compile
Set Of Instructions
Python Virtual Machine
a
b
4
c
2
6
Byte Code
PVM
Output
Machine Code

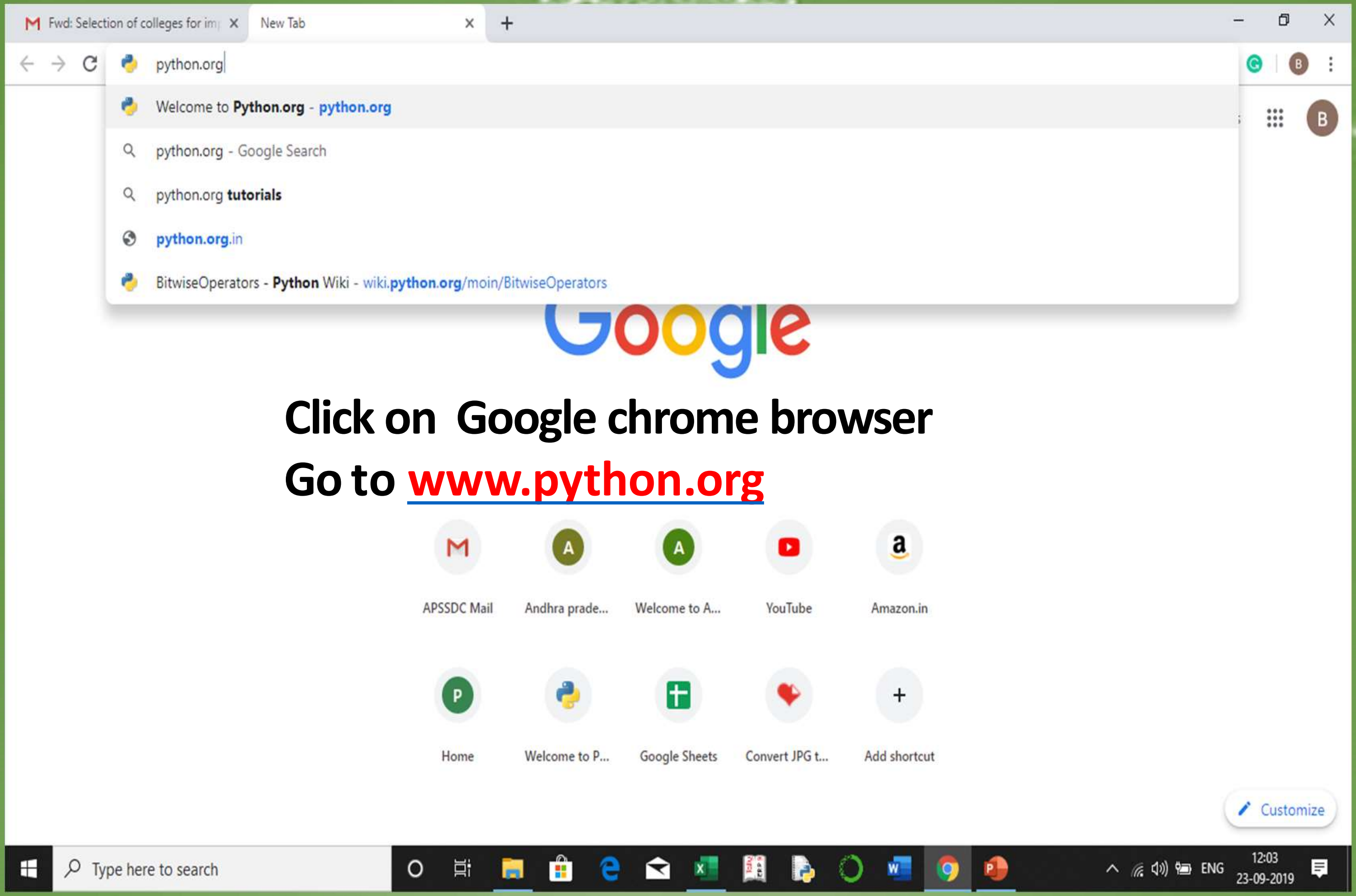
INVENTION OF PYTHON



- Guido van Rossum
- **Python** was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC **language** (itself inspired by SETL), capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989.

HOW TO DOWNLOAD AND INSTALL PYTHON

STEP : 1



python.org

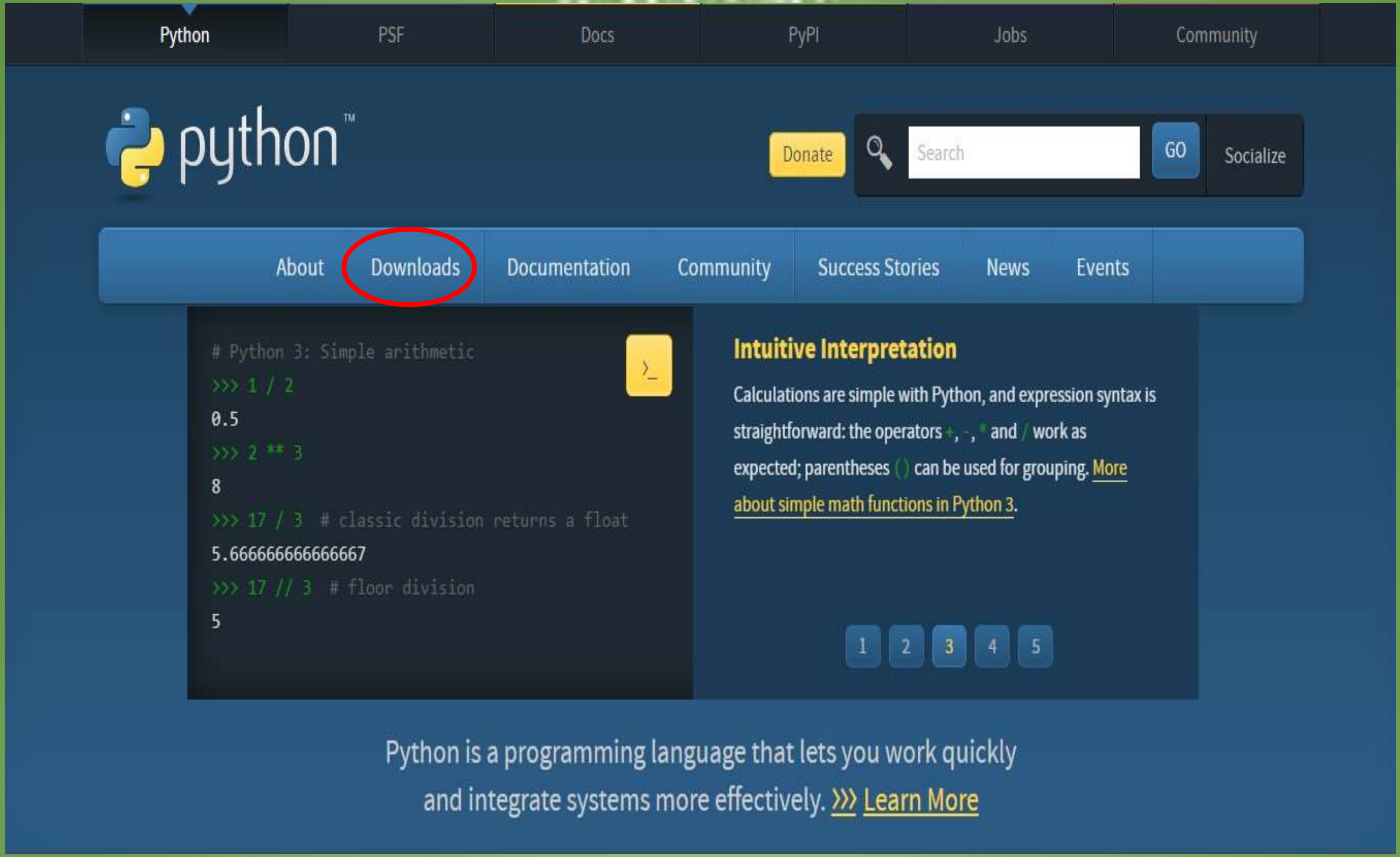
- Welcome to **Python.org** - **python.org**
- python.org - Google Search
- python.org **tutorials**
- **python.org**.in
- BitwiseOperators - **Python** Wiki - wiki.**python.org**/moin/BitwiseOperators

Click on Google chrome browser

Go to www.python.org

STEP : 2

Click “Downloads” Link at the top of the page



```
# Python 3: Simple arithmetic
>>> 1 / 2
0.5
>>> 2 ** 3
8
>>> 17 / 3  # classic division returns a float
5.666666666666667
>>> 17 // 3  # floor division
5
```

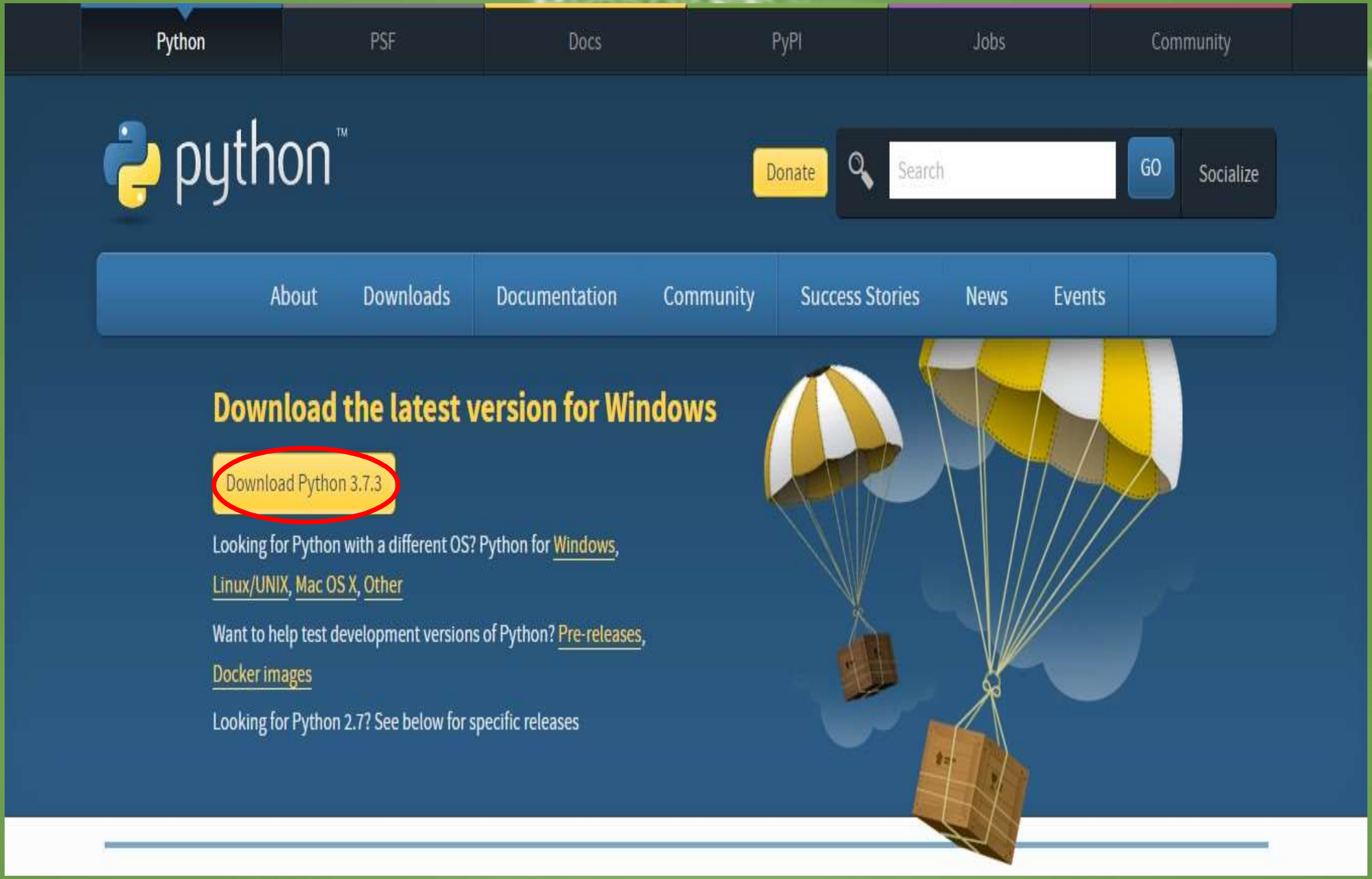
Intuitive Interpretation

Calculations are simple with Python, and expression syntax is straightforward: the operators +, -, * and / work as expected; parentheses () can be used for grouping. More about simple math functions in Python 3.

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

STEP : 3

- Click "Downloads" Link at the top of the page
- Click on "Download Python 3.7.3"



Download the latest version for Windows

Download Python 3.7.3

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases, Docker images

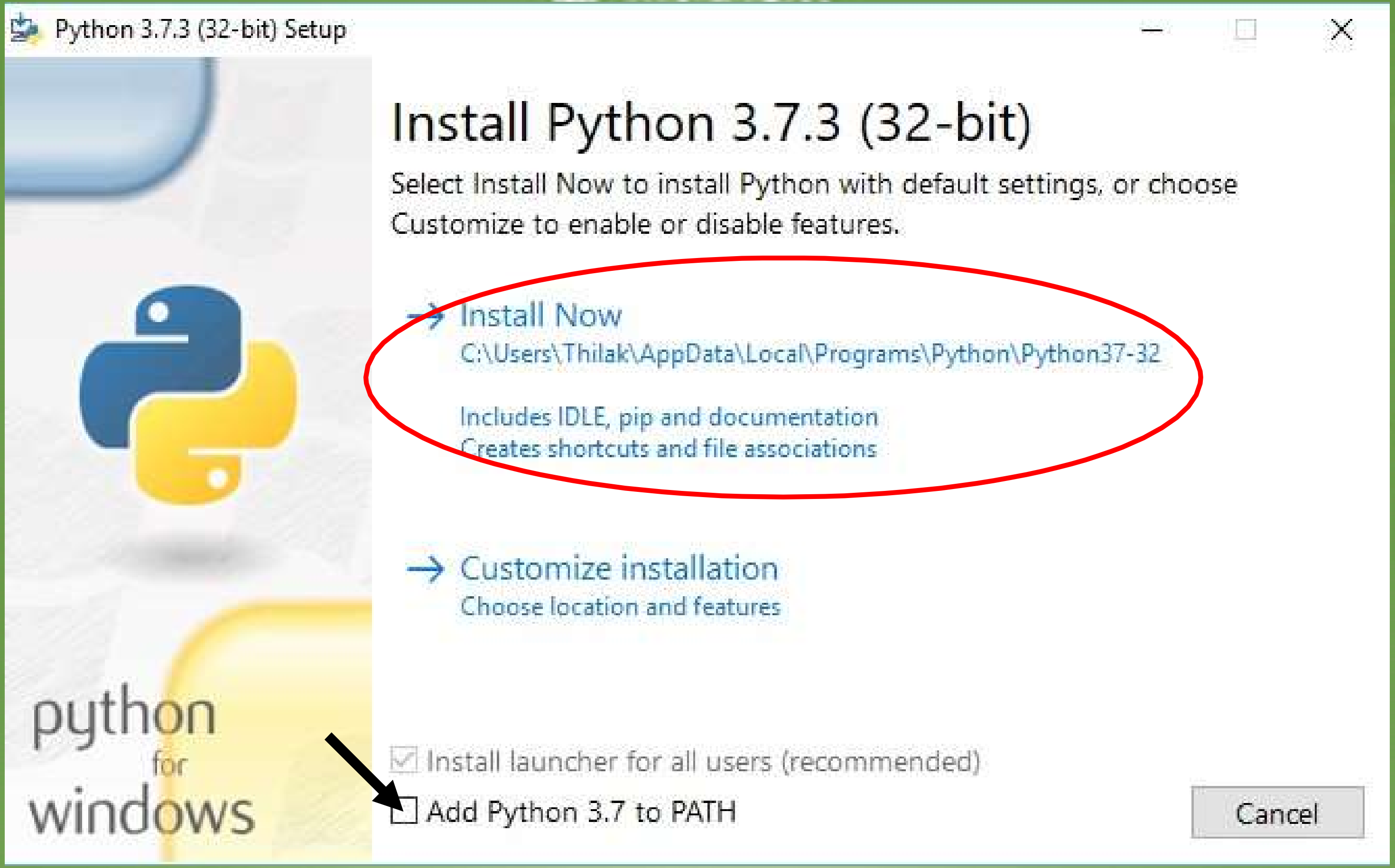
Looking for Python 2.7? See below for specific releases

STEP : 4

When the installation window comes up, click "Install Now"

- You can choose to "Add Python 3.7.3 to PATH"
- Note: Depending on how Windows is set up, you might need to provide an administrator password to install on your system at this point.
- You can choose to "Customize Installation" if you want, especially if you want to install to a location other than the default one shown. Generally I recommend installing to the default location unless you have a problem doing so.
- In any case, you might want to note the location of the installation in case you have difficulty later. If you are specifying the location yourself, put it in a location you are likely to easily find/remember.

STEP : 5



Python 3.7.3 (32-bit) Setup

Install Python 3.7.3 (32-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

→ Install Now
C:\Users\Thilak\AppData\Local\Programs\Python\Python37-32

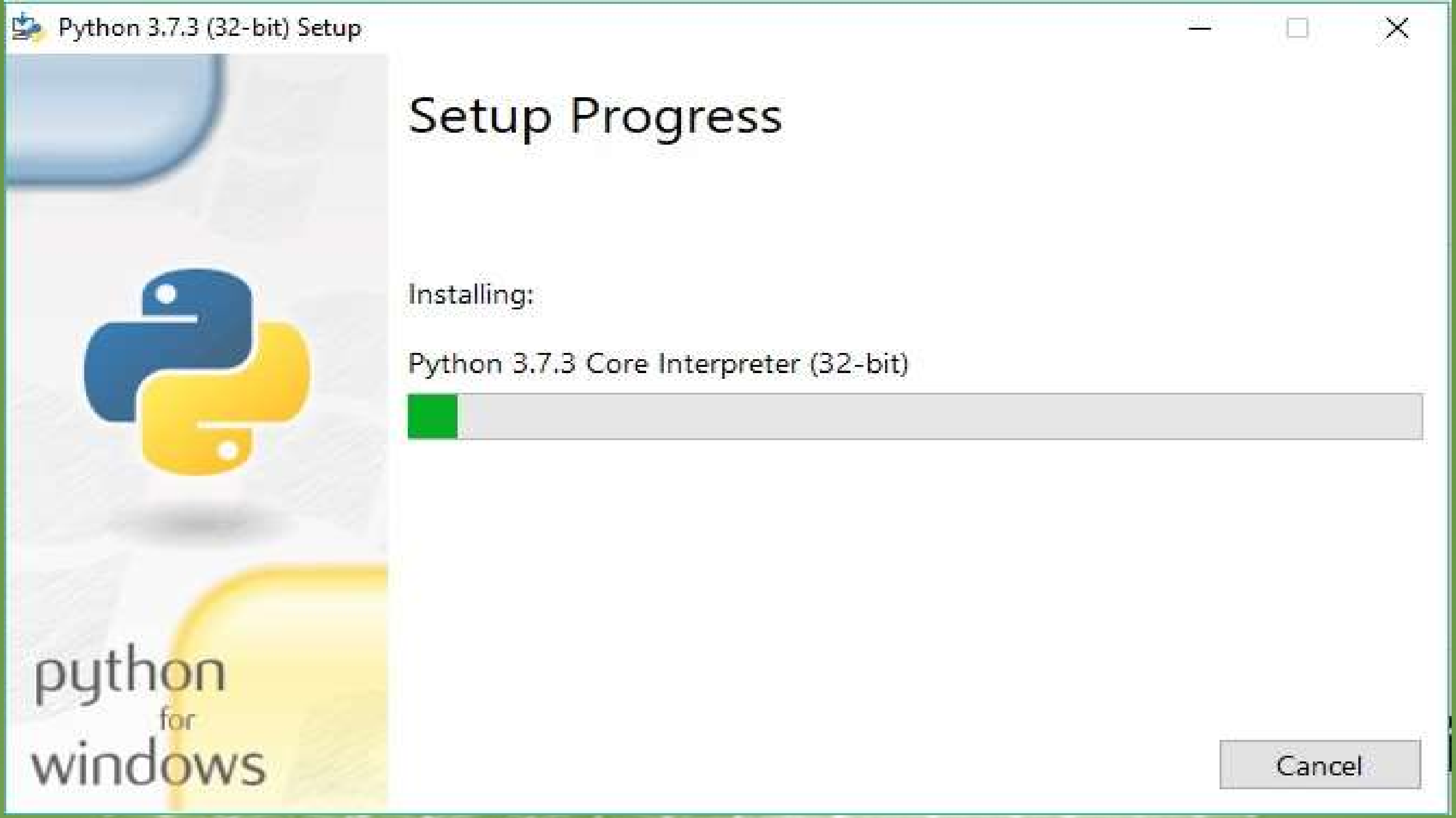
Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)

□ Add Python 3.7 to PATH

STEP : 6



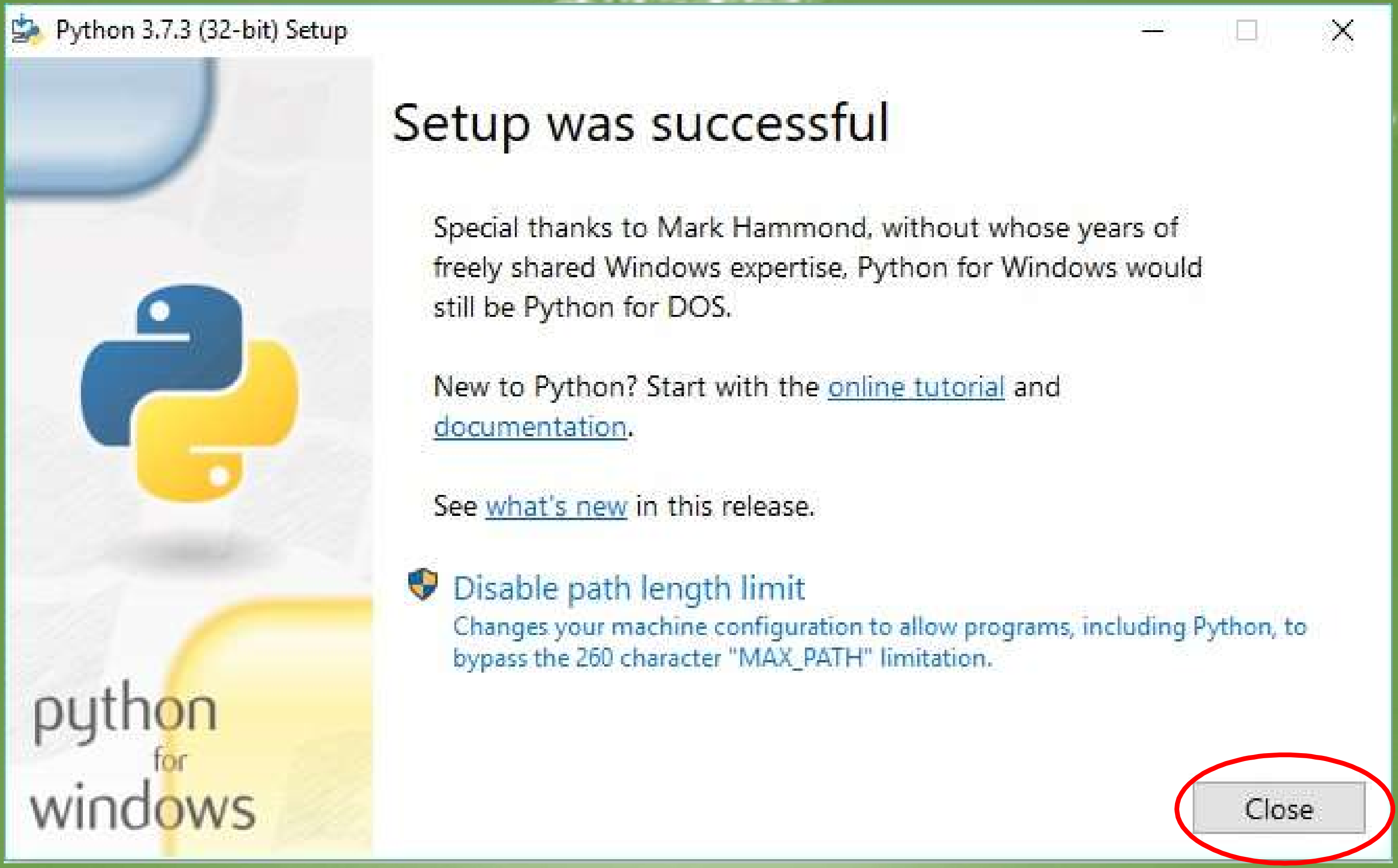
Python 3.7.3 (32-bit) Setup

Setup Progress

Installing:

Python 3.7.3 Core Interpreter (32-bit)

STEP : 7



Python 3.7.3 (32-bit) Setup

Setup was successful

Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

New to Python? Start with the online tutorial and documentation.

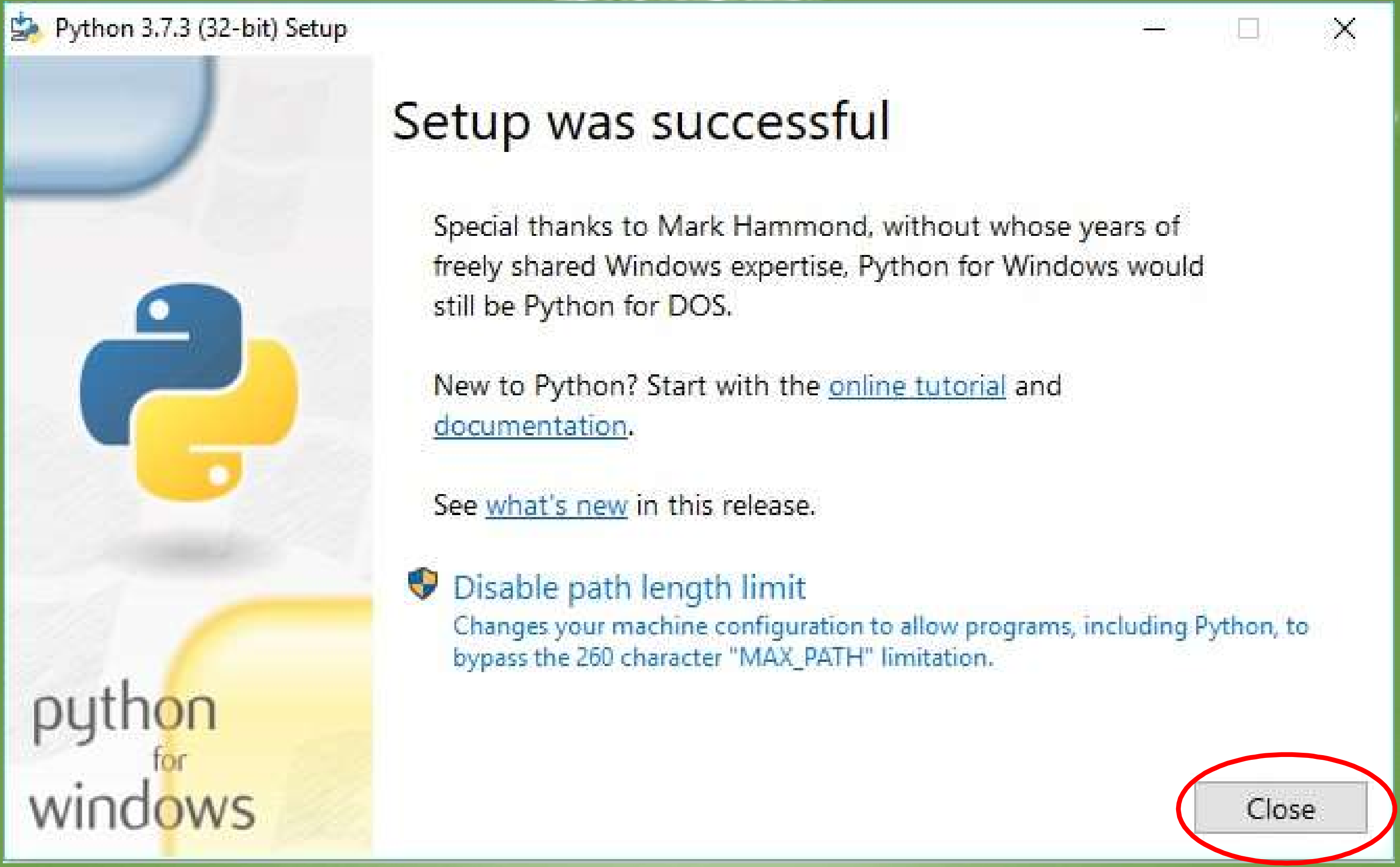
See what's new in this release.

Disable path length limit

Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

- You should see Python installing at this point.
- When it finishes, you should see a screen that says the installation was successful.

STEP : 7



Python 3.7.3 (32-bit) Setup

Setup was successful

Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

New to Python? Start with the online tutorial and documentation.

See what's new in this release.

Disable path length limit

Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

- You can click “Close”

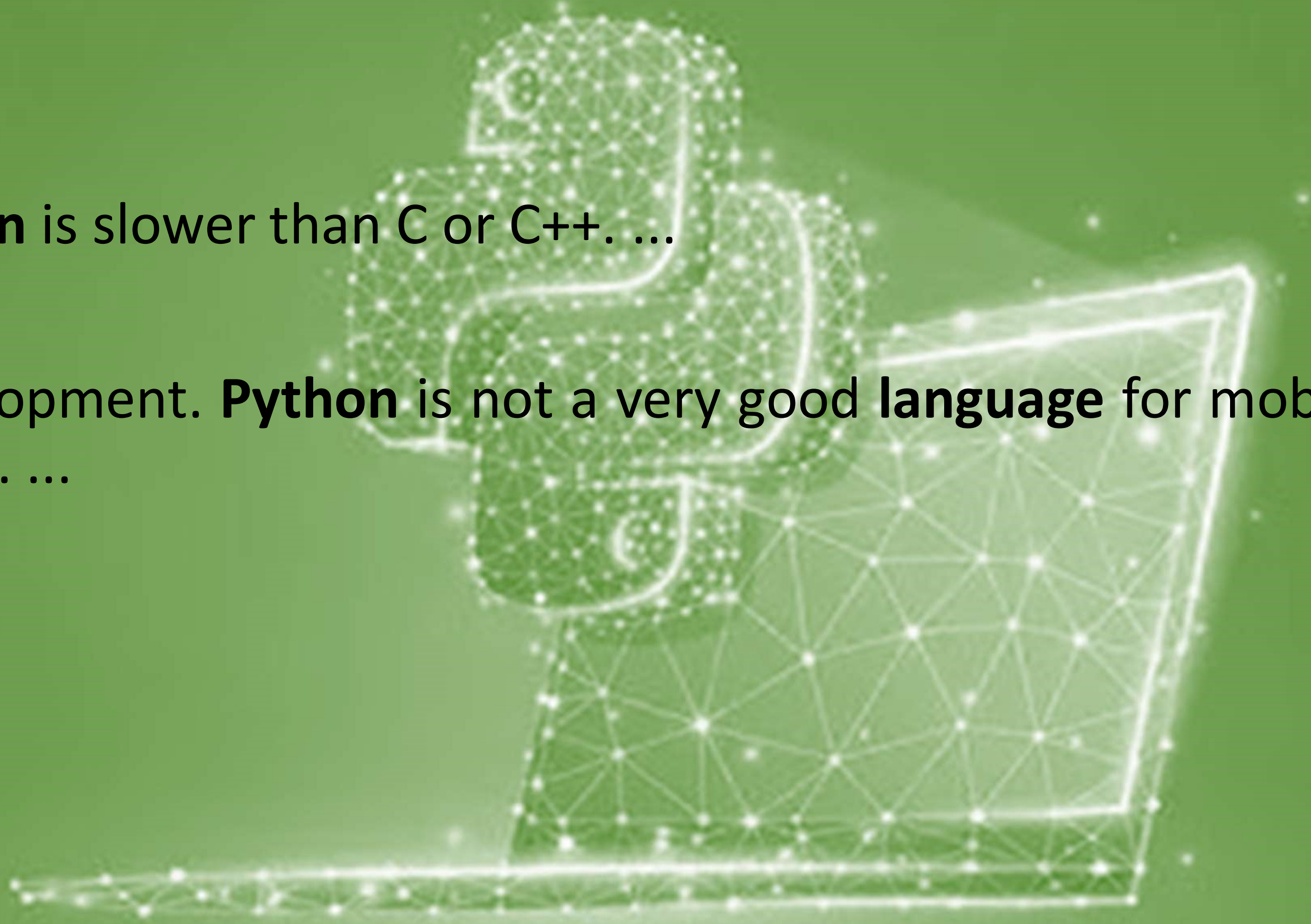
APPLICATIONS OF PYTHON

1) **Web applications**
2) **Image based applications**
3) **Internet of things**
4) **Cad based applications**
5) **Enterprise applications**
6) **Artificial intelligence**
7) **Machine learning**
8) **GUI based desktop applications(Games, Scientific Applications)**
9) **Operating Systems**
10) **Language Development**
11) **Prototyping**

Companies
used
P
Y
T
H
O
N

1) **Google(Components of Google spider and Search Engine)**
2) **Yahoo(Maps)**
3) **YouTube**
4) **Mozilla**
5) **Dropbox**
6) **Microsoft**
7) **Cisco**
8) **Spotify**
9) **Quora**

DISADVANTAGE OF PYTHON



- Speed. **Python** is slower than C or C++. ...
- Mobile Development. **Python** is not a very good **language** for mobile development . ...

SYNTAX OF ‘PYTHON’ LANGUAGE

- Now, writing the same program in PYTHON programming language:

```
print(“Andhra Pradesh State Skill Development Corporation”)
```

Output :

Andhra Pradesh State Skill Development Corporation

Comment Lines

- There are 2 types of comment lines in Python.
  1. Single line comment line

     #Python is a High level Programming language

  2. Multiline comment line

```
“””Python is a
   high level
   programming language
“””
```

Escape Sequences

- ➢These escape sequences starts with Backslash

S No	Backslash character	Name	Meaning
1	“\n”	New line character	Goes to New line
2	“\t”	Tabular Space	It gives Space
3	“\b”	Back space	Moves to Previous space
4	“\r”	Carriage return	Carriage returns
5	“\a”	Alarm	Beep sound
6	“\\”	Back slash	It gives single Backslash
7	“\””	Double quote	It gives Double quote

1. What are Identifiers and Variables

An **identifier** is a string of alphanumeric characters that begins with an alphabetic character or an underscore character that are used to represent programming elements such as variables, functions, Sequences(Lists and tuple … in python), and so on.

- ➢ An **identifier** is a user-defined word.

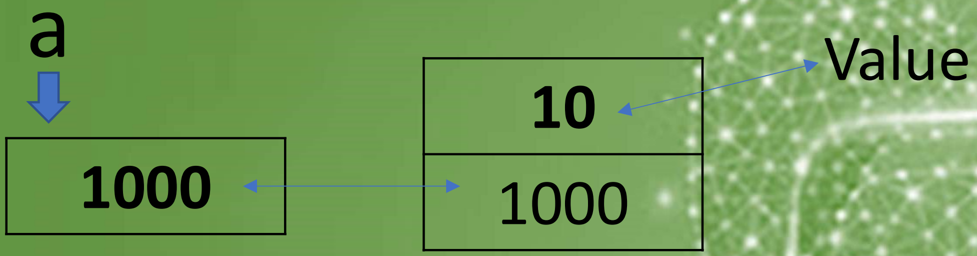
Identifier

Variable

Variable is an Identifier that occupies some part of Memory which can hold only one value.

- ➢It is not possible for a variable to hold more than 1 value.

- A variable contains two parts, a **value**, and a **symbolic name**.
- In Python



a

1000

10

1000

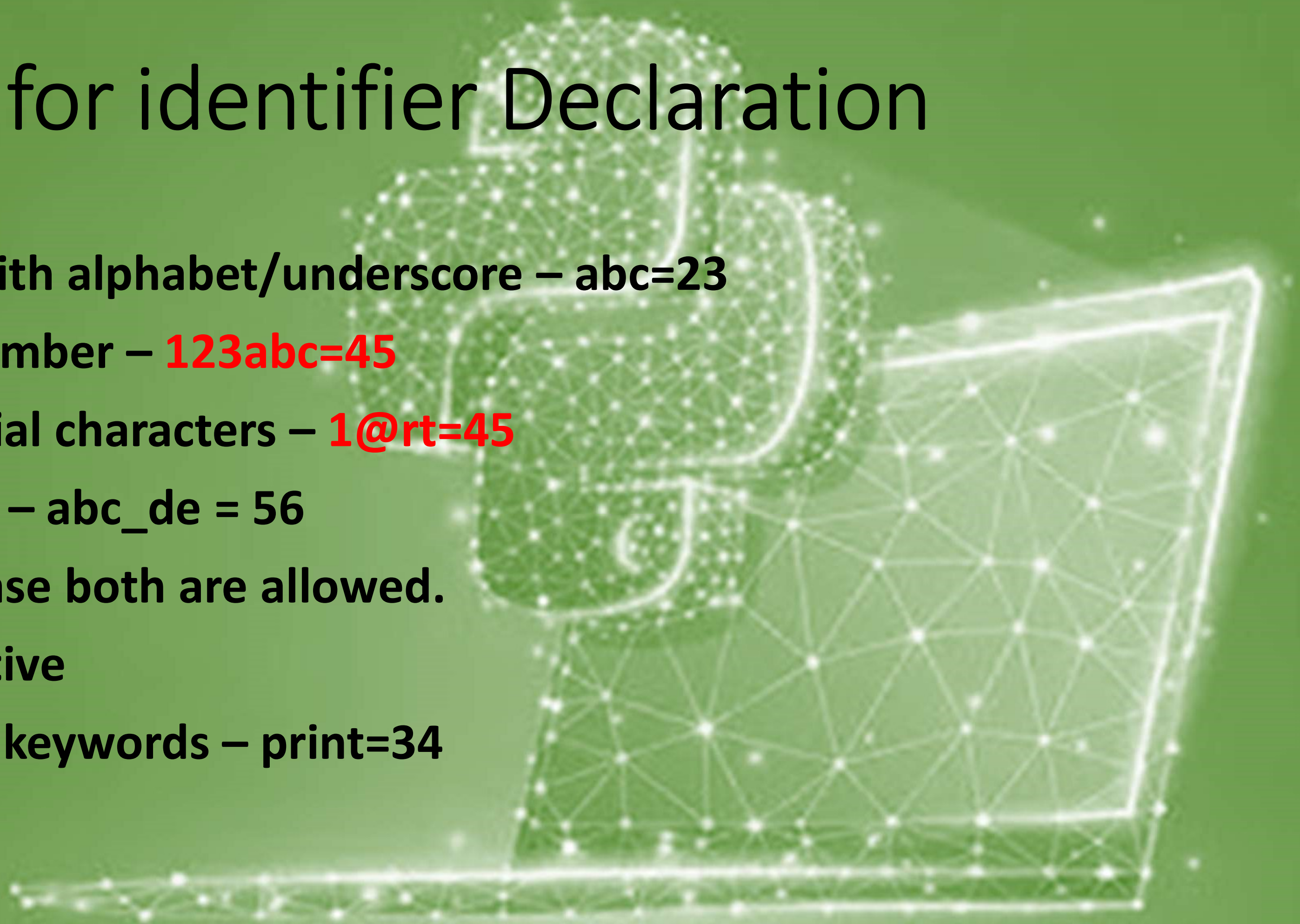
Value

- Declaring Variables:

Python is a dynamic language

```
a=10 #integer
b="venu"     #string
c=[" venu","ESC Coordinator", "in","APSSDC"]   #list
```

Rules for identifier Declaration



- **Identifier must start with alphabet/underscore – abc=23**
- **Must not start with number – 123abc=45**
- **Must not contain special characters – 1@rt=45**
- **Only allow underscore – abc_de = 56**
- **Upper case or lower case both are allowed.**
- **Identifier is case sensitive**
- **Don’t give reserved or keywords – print=34**

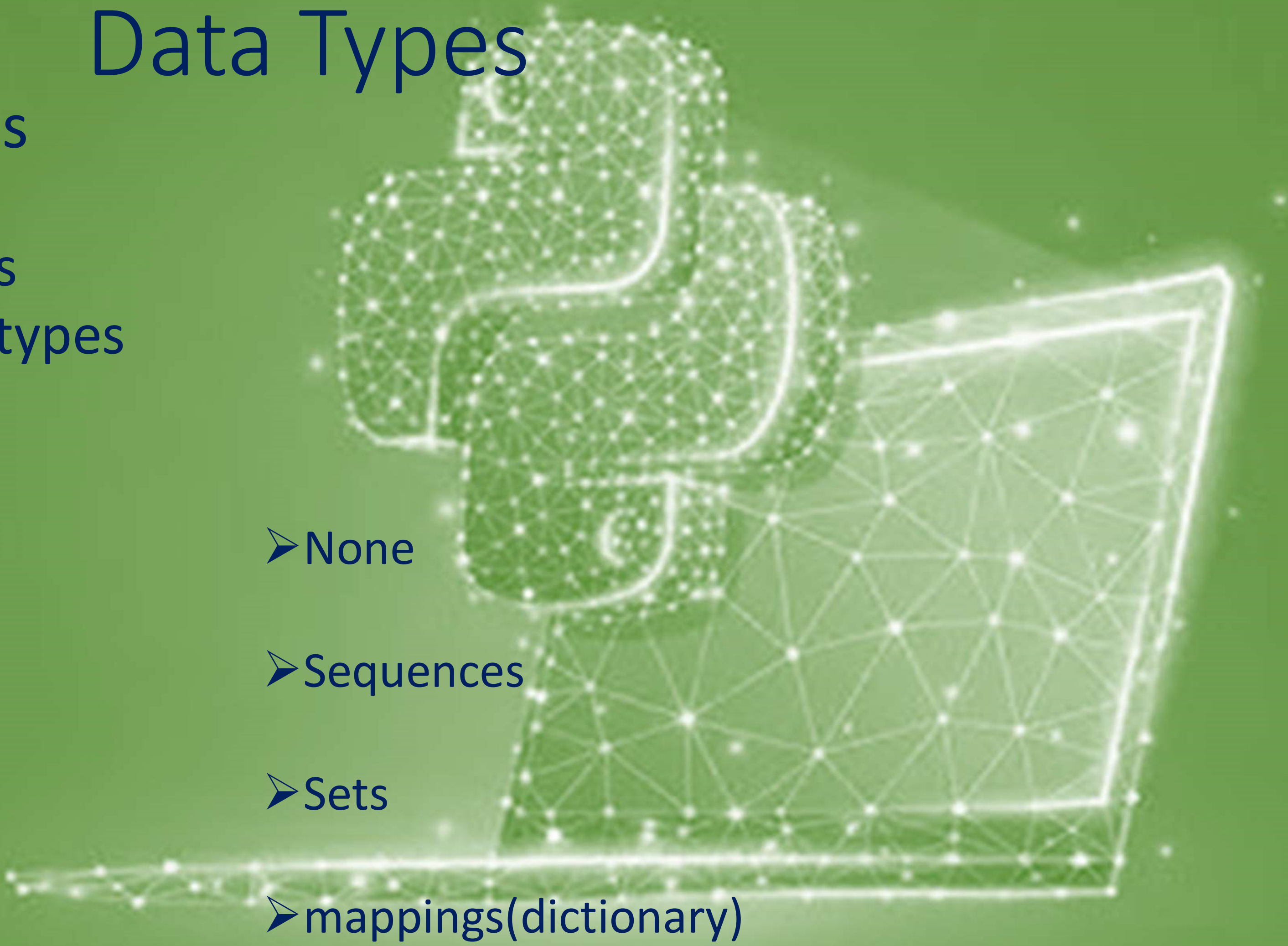
Data Types

- Types of data types
  1. Built in data types
  2. Derived data types
  3. User defined datatypes

1.Built in data types

- Int
- Float
- complex
- bool

- None
- Sequences
- Sets
- mappings(dictionary)



1.Int data type

- It represents only integer values either Positive or Negative

  i)a=10

  ii)b=-11

2.float data type

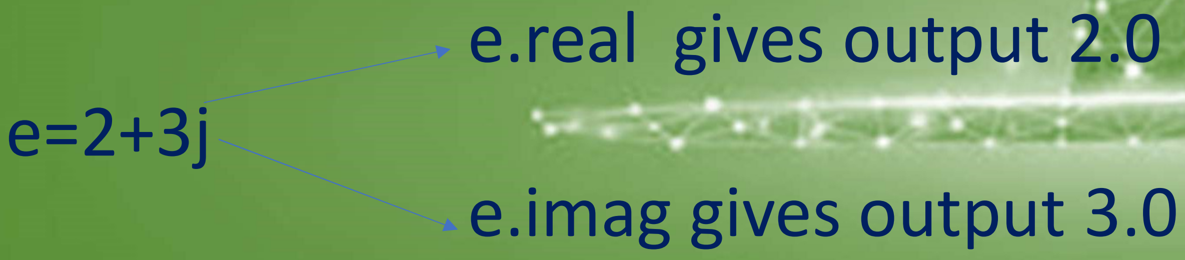
- It represents float(decimal) values either Positive or Negative

  i)c=3.56

  ii)d=-3.67

3.complex datatype

- It represents complex values (a+bj form) where a is real value and b is imaginary value.



e=2+3j

e.real gives output 2.0

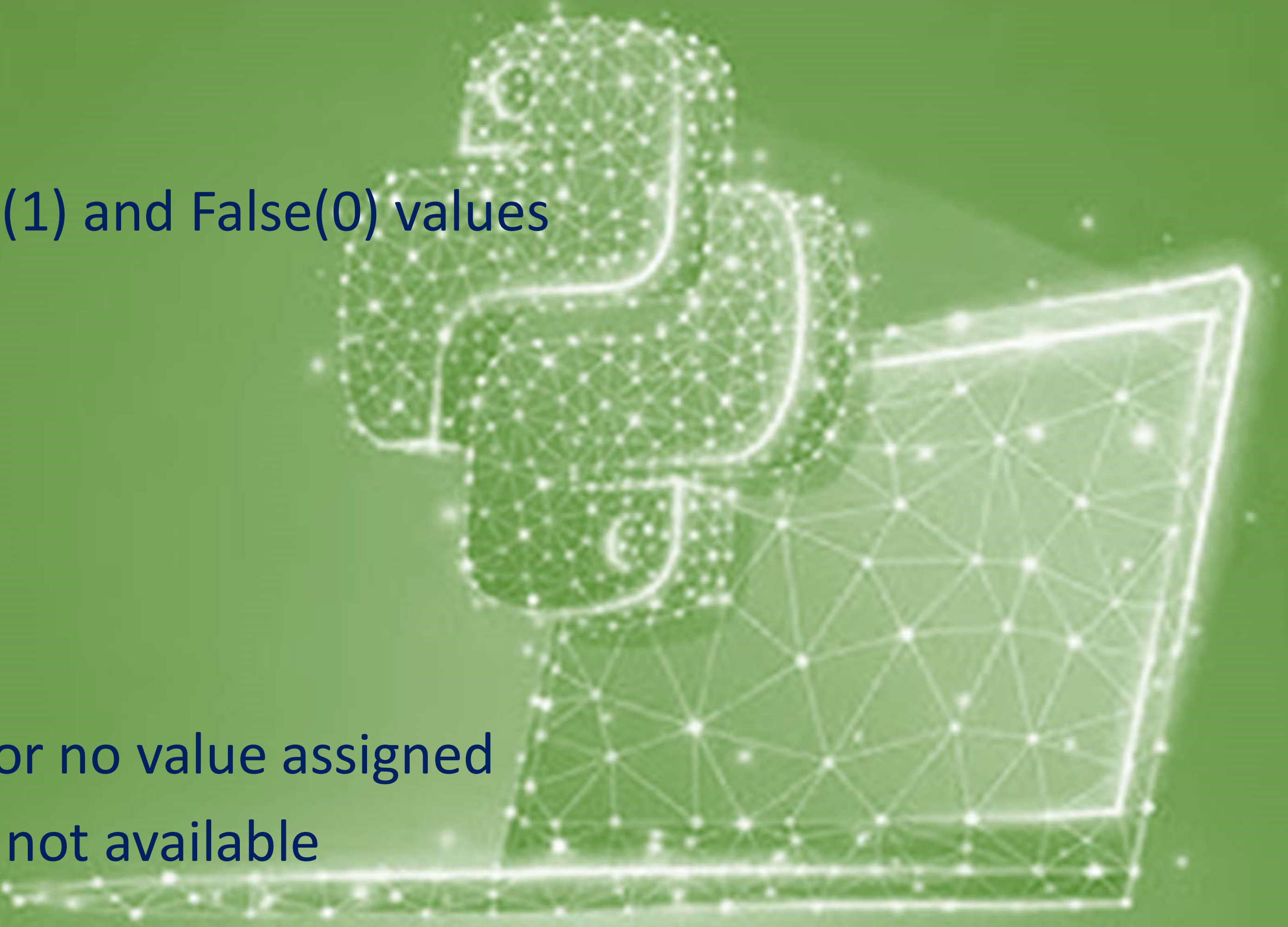
e.imag gives output 3.0

4. bool data type

- It represents only True(1) and False(0) values
  - i) f=True
  - ii)g=False

5. None data type

- It represents nothing or no value assigned
- It is used when data is not available



5.Sequences

1.String

Any number or text or special character which are enclosed in single quotes(‘string‘) or double quotes(“string“).

Syntax: variable=“anything”

i) a=“prakasam”

ii)b=“1234567@”

Index concept

Possitive index	0	1	2	3	4	5	6	7
	p	**r**	**a**	**k**	**a**	**s**	**a**	**m**
Negative index	-8	-7	-6	-5	-4	-3	-2	-1

Operators on strings

1. Concatenation(+):

Ex:
“venu”+”gopal”=“venugopal”

2. Repetition(*):

Ex:“venu”*3=“venuvenuvenu”

3. Slicing([]):

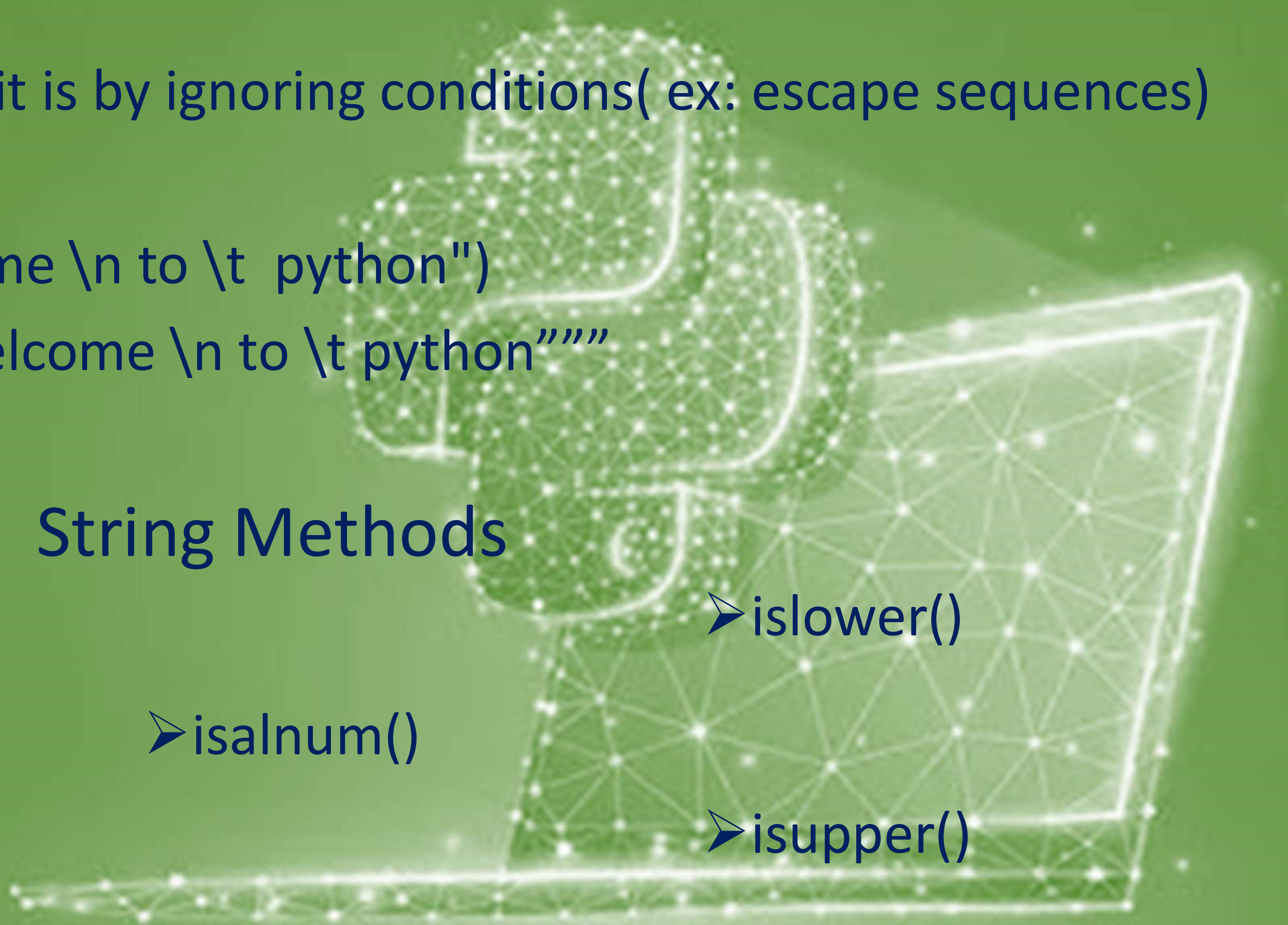
```
a="venugopal"
print(a[0:4])         #venu
print(a[4:])          #gopal
print(a[:])           #Venugopal
Print(a[1:-3])        #enugo
print(a[0:7:2])       #vngp
print(a[-1:-8:-2])    #lpgn
print(a[0::3])        #vup
print(a[:4:-1])       #lapo
print(a[::-1])        #lapogunev
print(a[::2])         #vngpl
print(a[:-1:-2])       #nothing
```

4. Raw string(r/R):

It prints string as it is by ignoring conditions(ex: escape sequences) present in it

example: print(r"welcome \n to \t python")

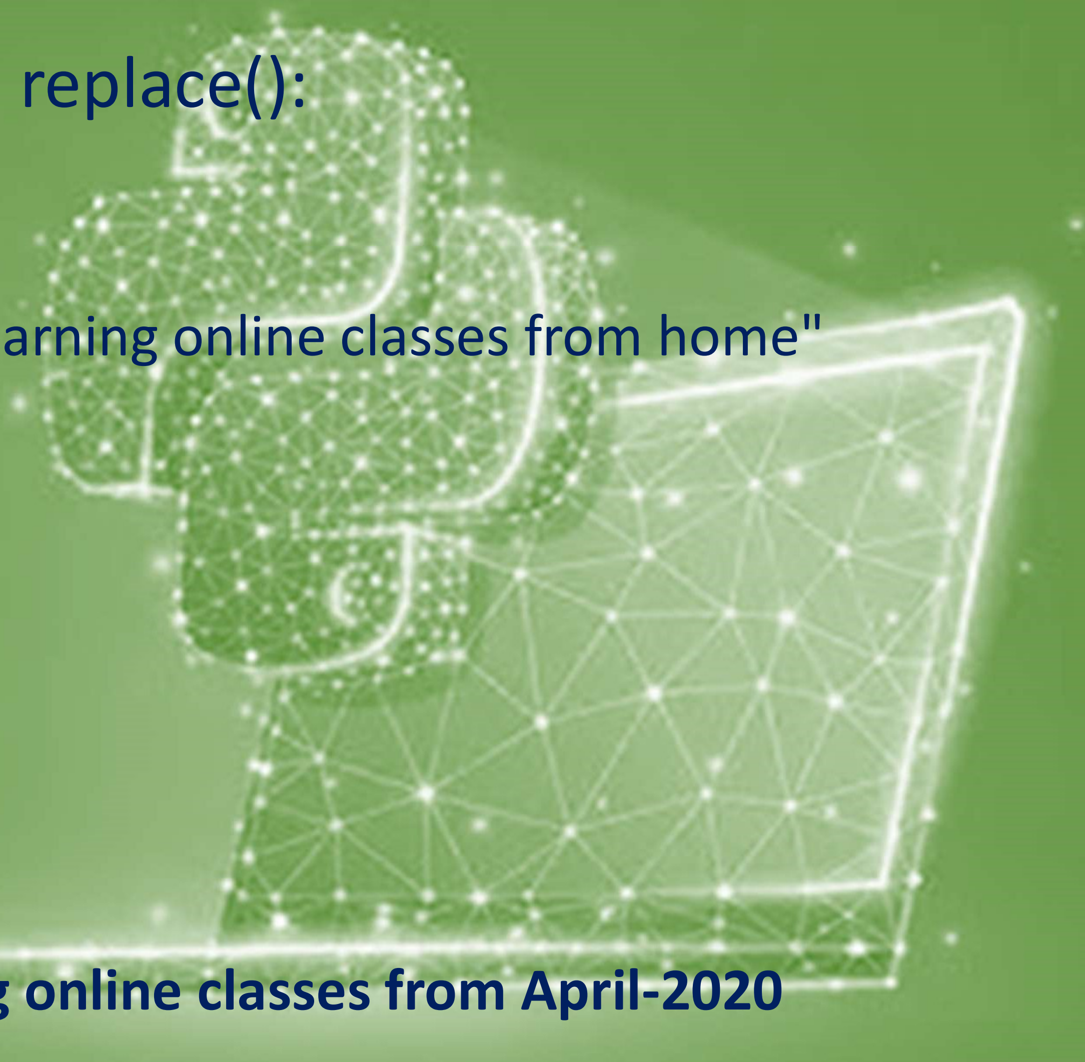
“”” output is welcome \n to \t python”””



String Methods

- capitalize()
- isalpha()
- isalnum()
- isdigit()
- islower()
- isupper()
- isspace()

➢join()   replace():



Example:

- a1= "because of corona virus we are learning online classes from home"
- a2=[" April","2020"]
- a3="-"
- a3=a3.join(a2)
- a1=a1.replace(“home”,a3)
- print(a1)

“””Output:

because of corona virus we are learning online classes from April-2020

“”””

List(collection of elements of different data types enclosed in [])

Syntax: variable=[ele1,ele2…….elen]

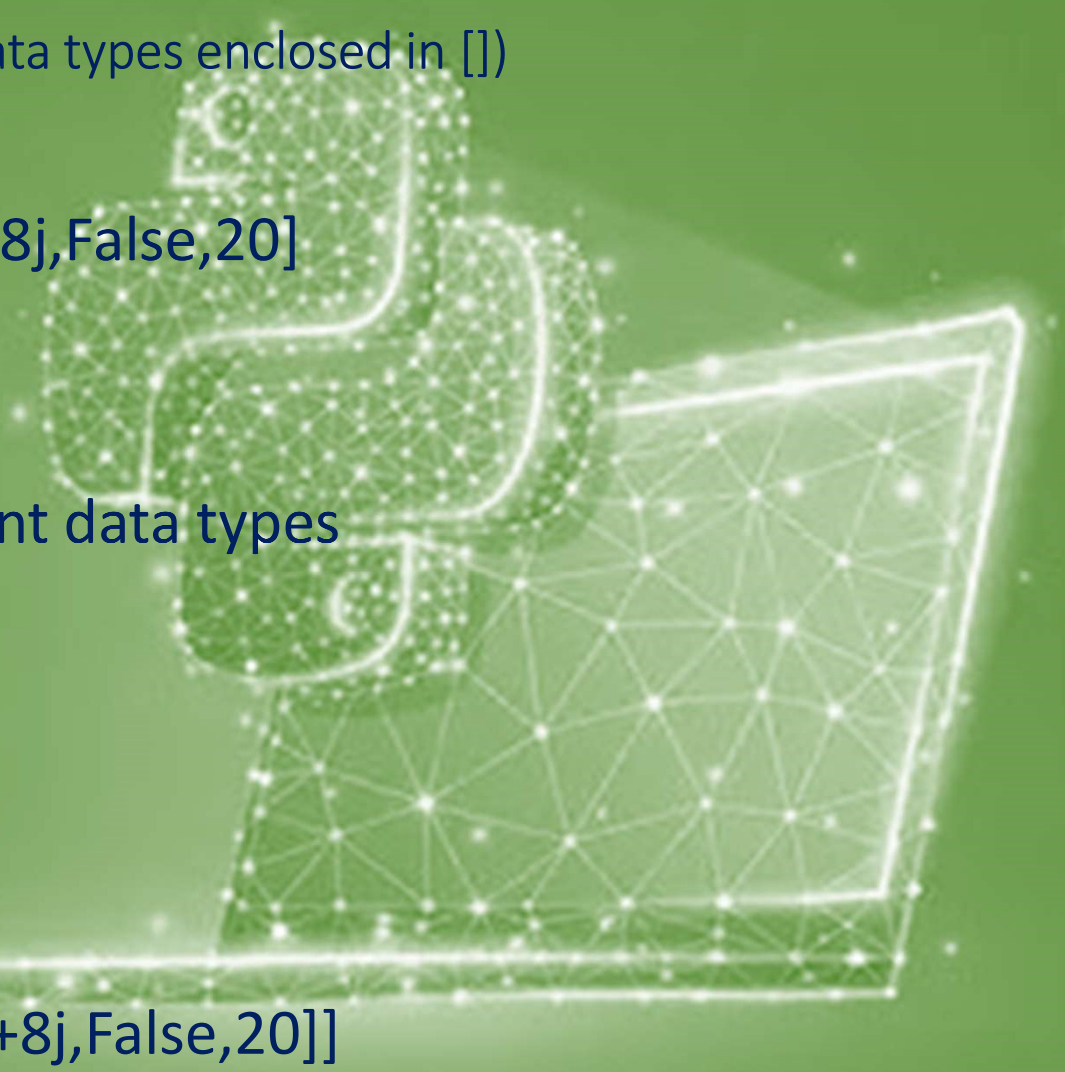
Ex: list=[20,2.5,“prakasam",2+8j,False,20]

- list is ordered
- List consists of elements of different data types
- List allows duplicate elements.

Nested list:

List in the list is called nested list

Ex: list=[20,2.5,“prakasam",[2+8j,False,20]]



- List is Mutable i.e we can modify the list.

Ex: list=[20,2.5,“prakasam",2+8j,False,20]

1. list.append(30) #list=[20,2.5,“prakasam",2+8j,False,20,30]
2. list.extend([40,”venu”]) #list=[20,2.5,“prakasam",2+8j,False,20,40,”venu”]
3. list.insert(1,10) #list=[20,10,“prakasam",2+8j,False,20]
4. list.remove(2.5) #list=[20,“prakasam",2+8j,False,20]
5. del list[2] #list=[20,2.5,2+8j,False,20]
6. del list #entire list will be deleted
7. list.pop(4) #pop will returns deleted element and list=[20,2.5,“prakasam",2+8j,20]

Methods of list

1. min()
2. max()
3. list.index(ele)
4. list.sort()
5. list.sort(key=None,reverse=False)
6. list.reverse()
7. list.sort(key=None,reverse=True)

tuple(collection of elements of different data types enclosed in ())

Syntax: variable=(ele1,ele2…….elen)

Ex: tuple=(20,2.5,“prakasam",2+8j,False,20)

- tuple is ordered
- tuple consists of elements of different data types
- tuple allows duplicate elements.
- Tuple is immutable. So we can’t modify the tuple.
- Deleting entire tuple

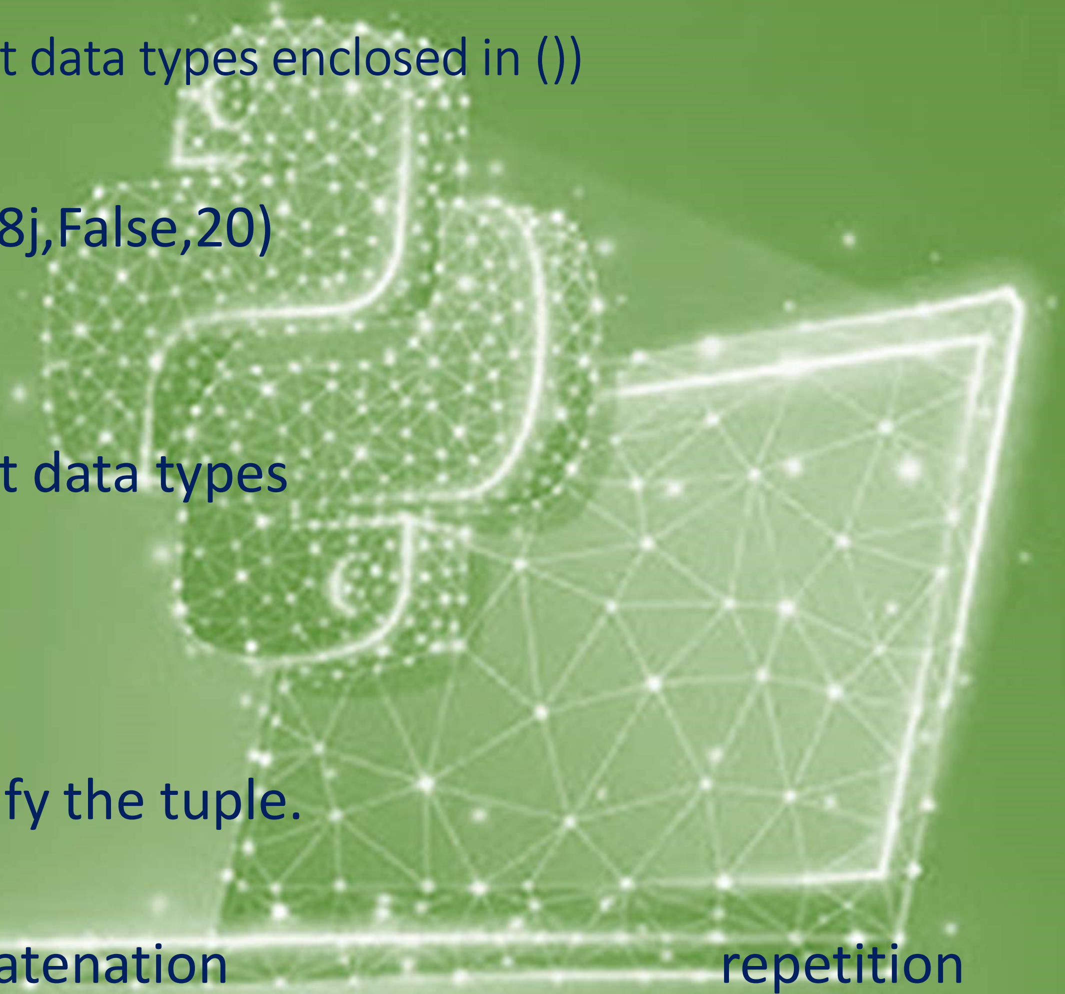
  del tuple

concatenation

(1,2,3)+(4,5,6)=(1,2,3,4,5,6)

repetition

(1,2,3)*2= (1,2,3,1,2,3)



Some technics to modify tuple

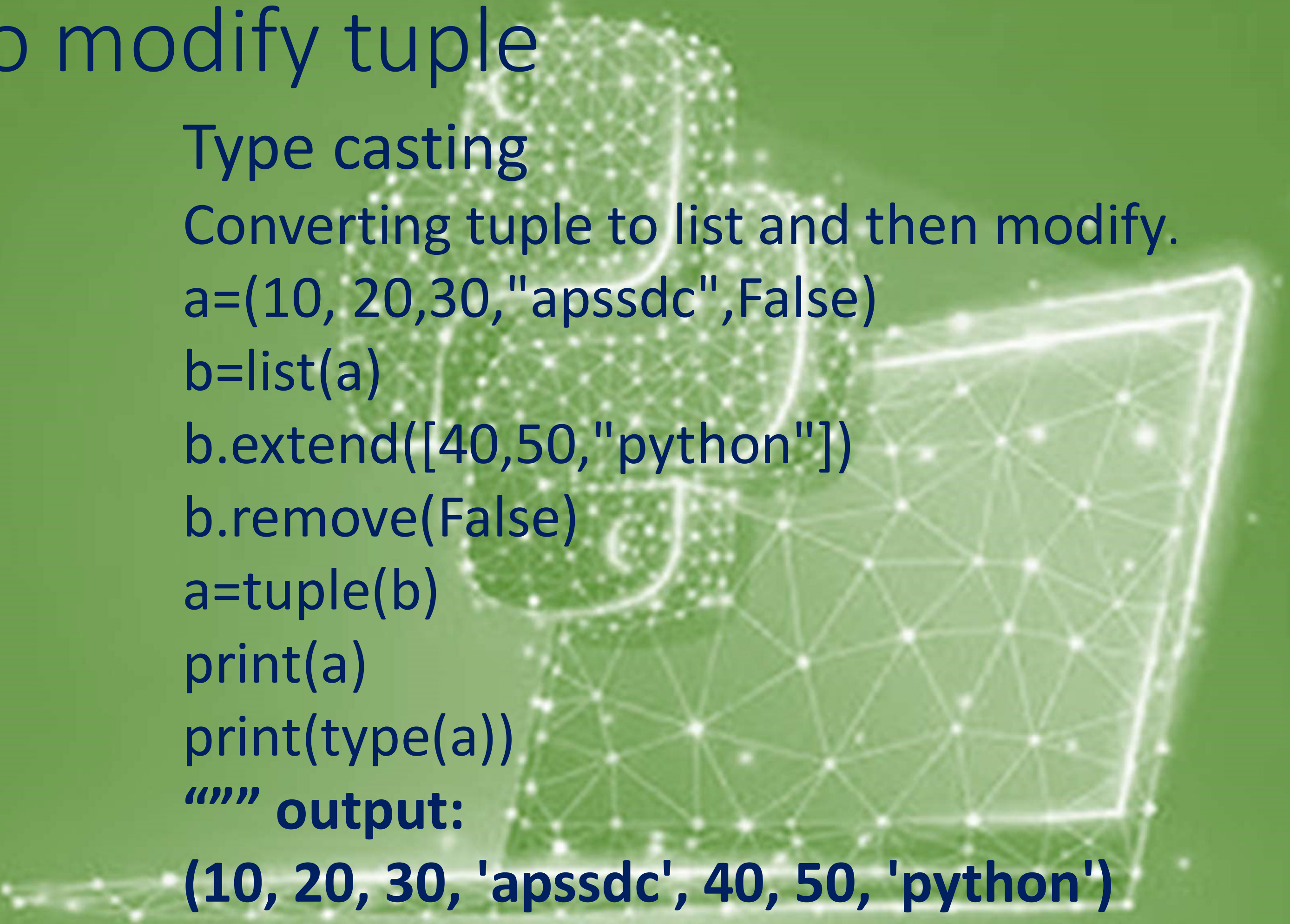
- Nested tuple

```
 a=(1,2,3,[4,5,6])
print(a)
a[3][1]=7
print(a)
“””output is
(1, 2, 3, [4, 7, 6])
“””
```

Type casting

Converting tuple to list and then modify.

```
a=(10, 20,30,"apssdc",False)
b=list(a)
b.extend([40,50,"python"])
b.remove(False)
a=tuple(b)
print(a)
print(type(a))
“”” output:
(10, 20, 30, 'apssdc', 40, 50, 'python')
<class 'tuple'>
“””
```



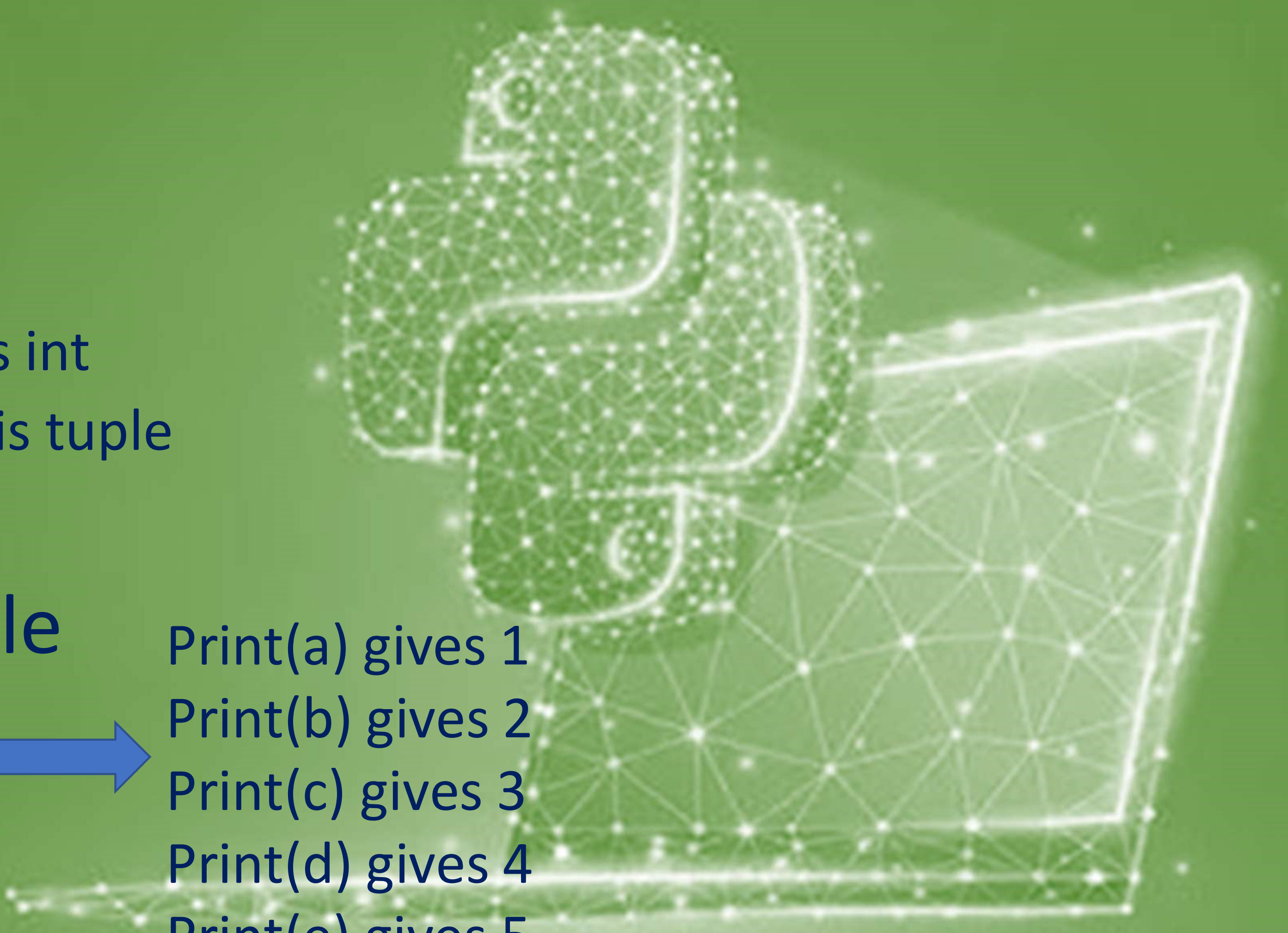
Packing tuple

a=1,2,3,4,5

type(a) is tuple

Note: type of b=(2) is int

type of b=(2,) is tuple



Unpacking tuple

tuple=1,2,3,4,5

a,b,c,d,e=tuple

→

Print(a) gives 1

Print(b) gives 2

Print(c) gives 3

Print(d) gives 4

Print(e) gives 5

Sets (collection of elements of different data types enclosed in {})

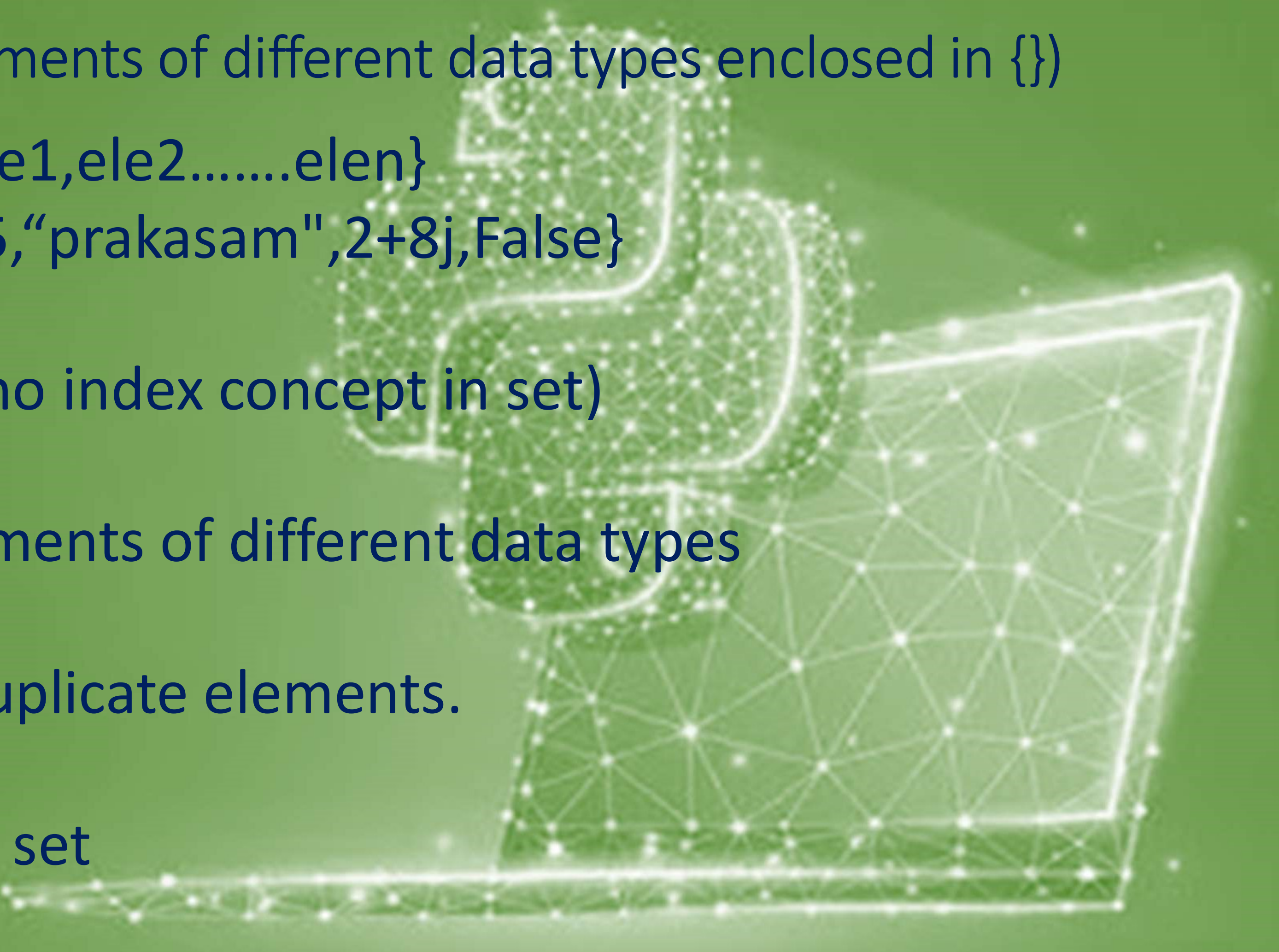
Syntax: variable={ele1,ele2…….elen}

```
Ex: set={20,2.5,“prakasam",2+8j,False}
```

- set is unordered(no index concept in set)
- set consists of elements of different data types
- set don’t allows duplicate elements.

Type casting list into set

```
list=[1,2,3,4]
s=set([1,2,3,4])
```



- Set is mutable

```
set={20,2.5,“prakasam",2+8j,False}
```

1. set.add(10)
2. set.remove(20)
3. set.discard(10)
4. del set

Difference between remove and discard

Remove():

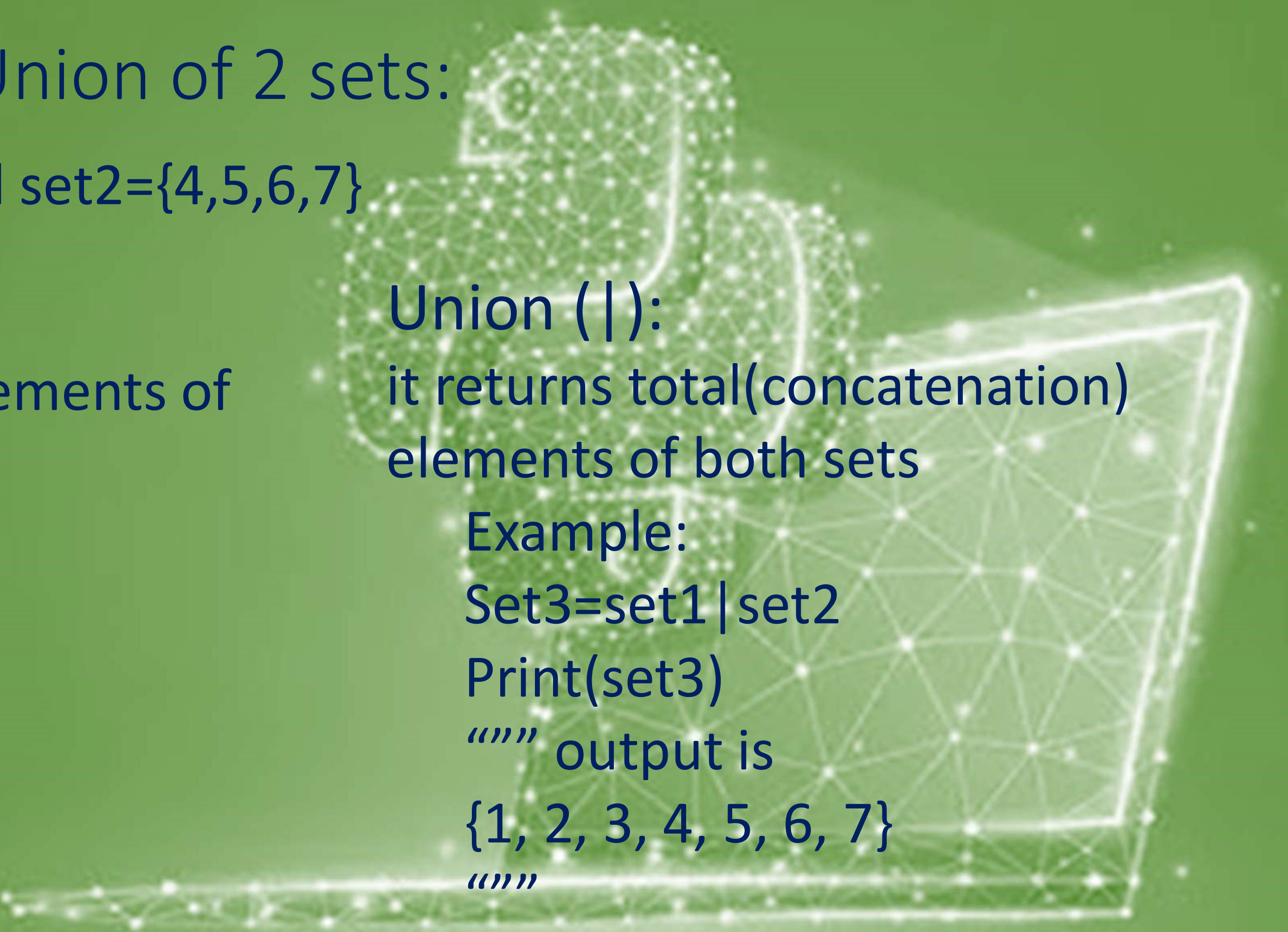
```
set={20,2.5,"prakasam",2+8j,False}
set.remove(50)
print(set)
“””output: Error
“””
```

Discard():

```
set={20,2.5,"prakasam",2+8j,False}
set.discard(50)
print(set)
“””output:
{False, 2.5, 'prakasam', 20, (2+8j)}
“””
```

Intersection and Union of 2 sets:

let set1={1,2,3,4} and set2={4,5,6,7}



Intersection(&):

it returns common elements of both sets

Example:

```
Set3=set1&set2
Print(set3)
“”” output is
{4}
“””
```

Union (|):

it returns total(concatenation) elements of both sets

Example:

```
Set3=set1|set2
Print(set3)
“”” output is
{1, 2, 3, 4, 5, 6, 7}
“””
```

frozenset

- It is same as set but frozenset is immutable.
- We can access set and frozenset valus by using for loop.

Creating frozenset:

```
        1. s={1,2,3,4}
        fzset=frozenset(s)
        2. fzset=frozenset({1,2,3,4}
```

Type casting:

```
        s={1,2,3,4}
        fzset=frozenset(s)
        s=set(fzset)
```

Range data type

- ➢range() is used to generate sequence of numbers.
- ➢Generally it is used to repeat for loop statements for number of times.

Example:

```
r=range(1,10,2)
for i in r:
   print(i,end=",")
“””output: 1,3,5,7,9”””
```

end keyword:

It is used to print 2 or more statements in a single line with any character.

```
Example: print("welcome",end=",")
         print("to python")
     “”” output : welcome,to python”””
```

Dictionary(mapping) data type

It is a collection of elements in the form of keys and values.

Syntax: variable={key1:value1,key2:value2…..keyn:valuen}

- Dictionary datatype can have different datatypes as elements.
- Dictionary datatype is unordered.
- In this datatype duplicate values allowed but duplicate keys not allowed
- Dictionary datatype is mutable

➢Modifying dictionary data type:

Example: ddt={10:”venu”,”a”:”gopal”,20:2.5}

1. ddt[30]=“new ele”   # ddt={10:”venu”,”a”:”gopal”,20:2.5,30:”new ele”}

2. ddt.update({40:2020})   # ddt={10:”venu”,”a”:”gopal”,20:2.5,40:2020}

3. del ddt[“a”]   # ddt={10:”venu”,20:2.5}

4. ddt.pop(20)   #ddt={10:”venu”,”a”:”gopal”}

5. ddt.values()   #dict_values([”venu”,”gopal”,2.5])

6. ddt.keys()   #dict_keys([10,”a”,20])

Bytes data type

- Bytes data type contains only bytenumbers where bytenumber is an positive number in the range (0 to 256(exclusive)).
- It is used to represents data(images, videos..etc) in binary form.
- Bytes datatype allows only int type it doesn't allow remaining datatypes as elements.
- Creating bytes datatype:

```
1. a=[1,2,3] # creating list
   b=bytes(a)
2. c=bytes([4,5,6])
```

- bytes datatype is immutable.

Bytearray data type

- Bytearray data type contains only bytenumbers where bytenumber is an positive number in the range (0 to 256(exclusive)).
- It is used to represents data(images, videos..etc) in binary form.
- Bytearray datatype allows only int type it doesn’t allow remaining datatypes as elements.
- Creating bytearray datatype:

  1. a=[1,2,3] # creating list

     b=bytearray(a)

  2. c=bytearray([4,5,6])

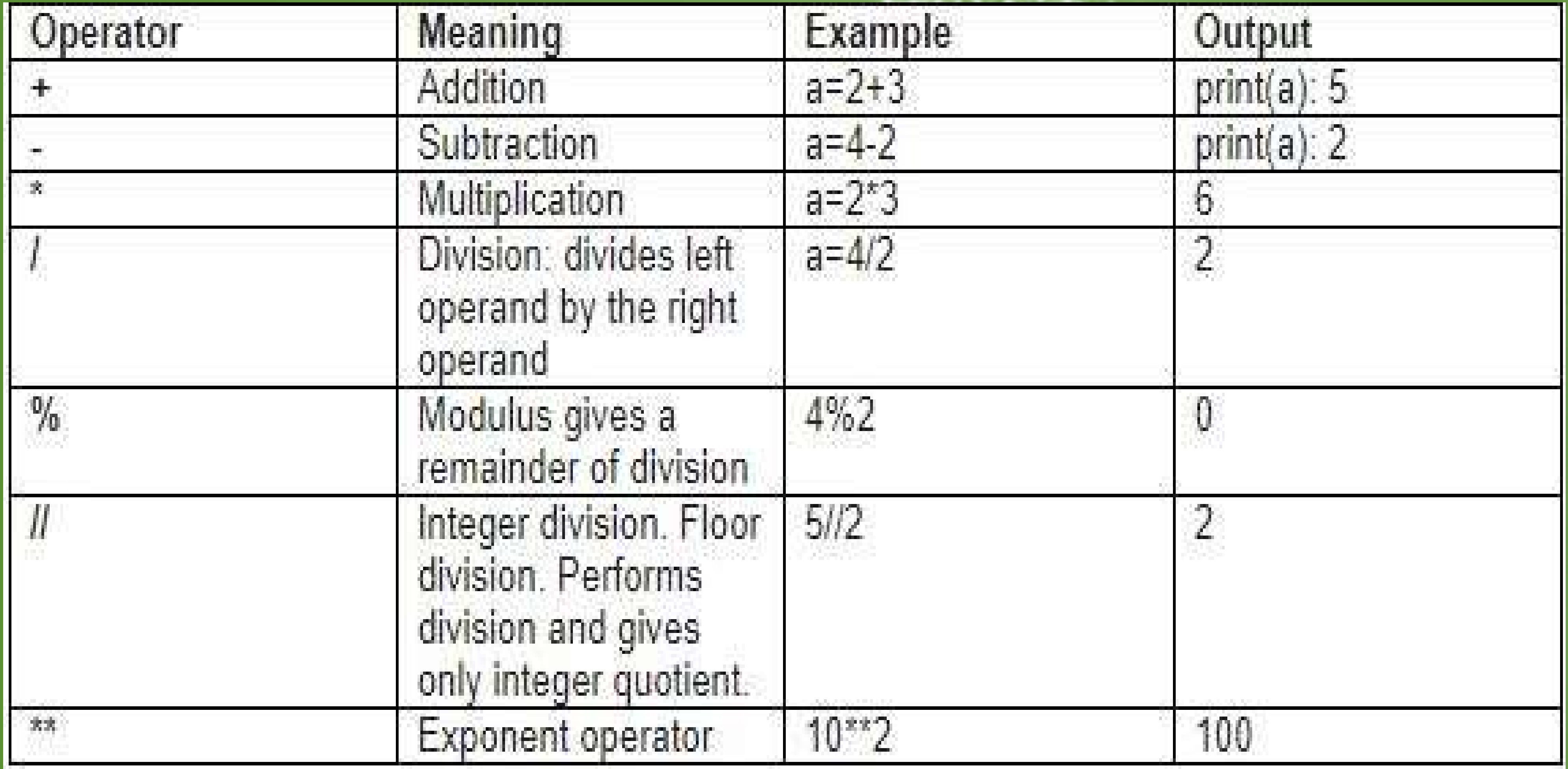
- bytearray datatype is mutable.

```
a=[1,2,3] # creating list
b=bytearray(a)
b[0]=4
for i in b:
   print(i,end=“ ”)
“””output : 4 2 3
```

Operators

1. Arithmetic operators
2. Relational operators
3. Assignment operators
4. Special (membership and identity) operators
5. Logical operators
6. Bitwise operators

Arithmetic operators



Operator	Meaning	Example	Output
+	Addition	a=2+3	print(a): 5
-	Subtraction	a=4-2	print(a): 2
*	Multiplication	a=2*3	6
/	Division: divides left operand by the right operand	a=4/2	2
%	Modulus gives a remainder of division	4%2	0
//	Integer division. Floor division. Performs division and gives only integer quotient.	5//2	2
**	Exponent operator	10**2	100

Arithmetic expression evaluation

- Preference of Arithmetic operators for evaluation

1st : ()

2nd: exponential(**)

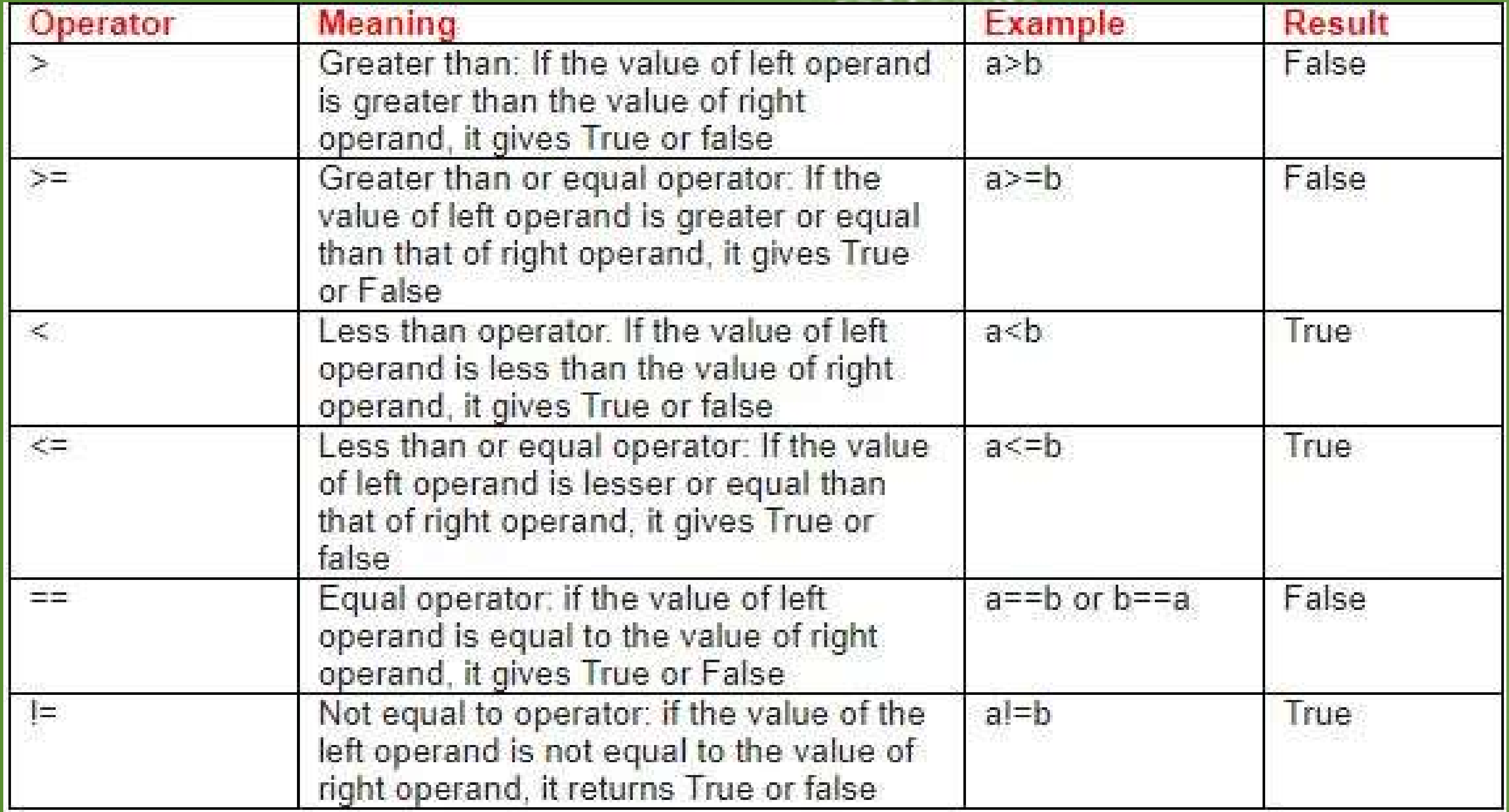
3rd: * (or) / (or) % (or) //

4th: + (or) –

Example →

```
5-6*4**2/8*3//(3+1)%2
5-6*4**2/8*3//4%2
5-6*16/8*3//4%2
 5-96/8*3//4%2
 5-12*3//4%2
 5-36//4%2
 5-9%2
 5-1
4.0
```

Relational operators



Operator	Meaning	Example	Result
>	Greater than: If the value of left operand is greater than the value of right operand, it gives True or false	a>b	False
>=	Greater than or equal operator: If the value of left operand is greater or equal than that of right operand, it gives True or False	a>=b	False
<	Less than operator: If the value of left operand is less than the value of right operand, it gives True or false	a<b	True
<=	Less than or equal operator: If the value of left operand is lesser or equal than that of right operand, it gives True or false	a<=b	True
==	Equal operator: if the value of left operand is equal to the value of right operand, it gives True or False	a==b or b==a	False
!=	Not equal to operator: if the value of the left operand is not equal to the value of right operand, it returns True or false	a!=b	True

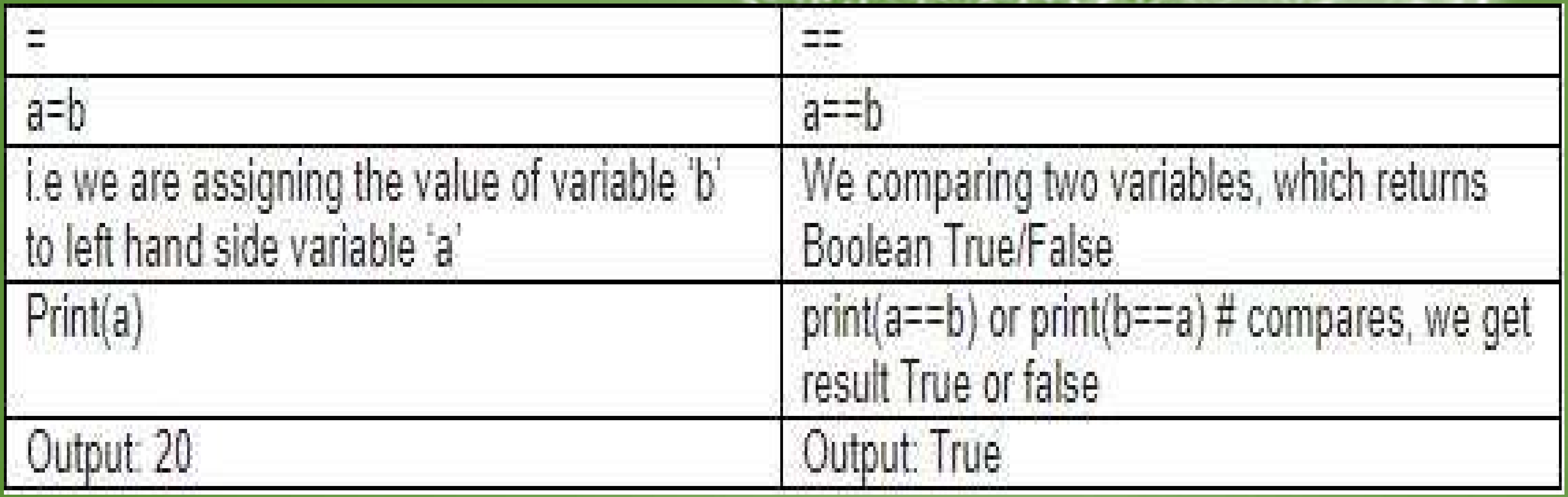
Example:

- a,b,c,d,h="10",20,10,"21.5“,5
- print("c+b=",c+b)
- print("a+d=",a+d)
- e=b>=c
- print("b>=c is",b>=c)
- print(1>=e)
- print(b>c<h)
- **ANS: c+b= 30 #adding integers**
- **a+d= 1021.5 # concatenation of strings**
- **b>=c is True # relational operator**
- **True # 1>=True is True**
- **False # chaining operation**

Assignment operators

- These are used to assign values to variable which is on left hand side

- Comparing between = and == operators



=	==
a=b	a==b
i.e we are assigning the value of variable 'b' to left hand side variable 'a'	We comparing two variables, which returns Boolean True/False
Print(a)	print(a==b) or print(b==a) # compares, we get result True or false
Output: 20	Output: True

Compound operators

1.	a+=b	a=a+b
2.	a-=b	a=a-b
3.	a*=b	a=a*b
4.	a/=b	a=a/b
5.	a%=b	a=a%b
6.	a//=b	a=a//b
7.	a**=b	a=a**b
8.	a&=b	a=a&b
9.	a|=b	a=a|b

Special (Identity and membership) operators

1. Identity operators:

i) is : a is b returns True if both a and b pointing to same object.

ii)is not: a is not b returns True if both a and b are not pointing to same object.

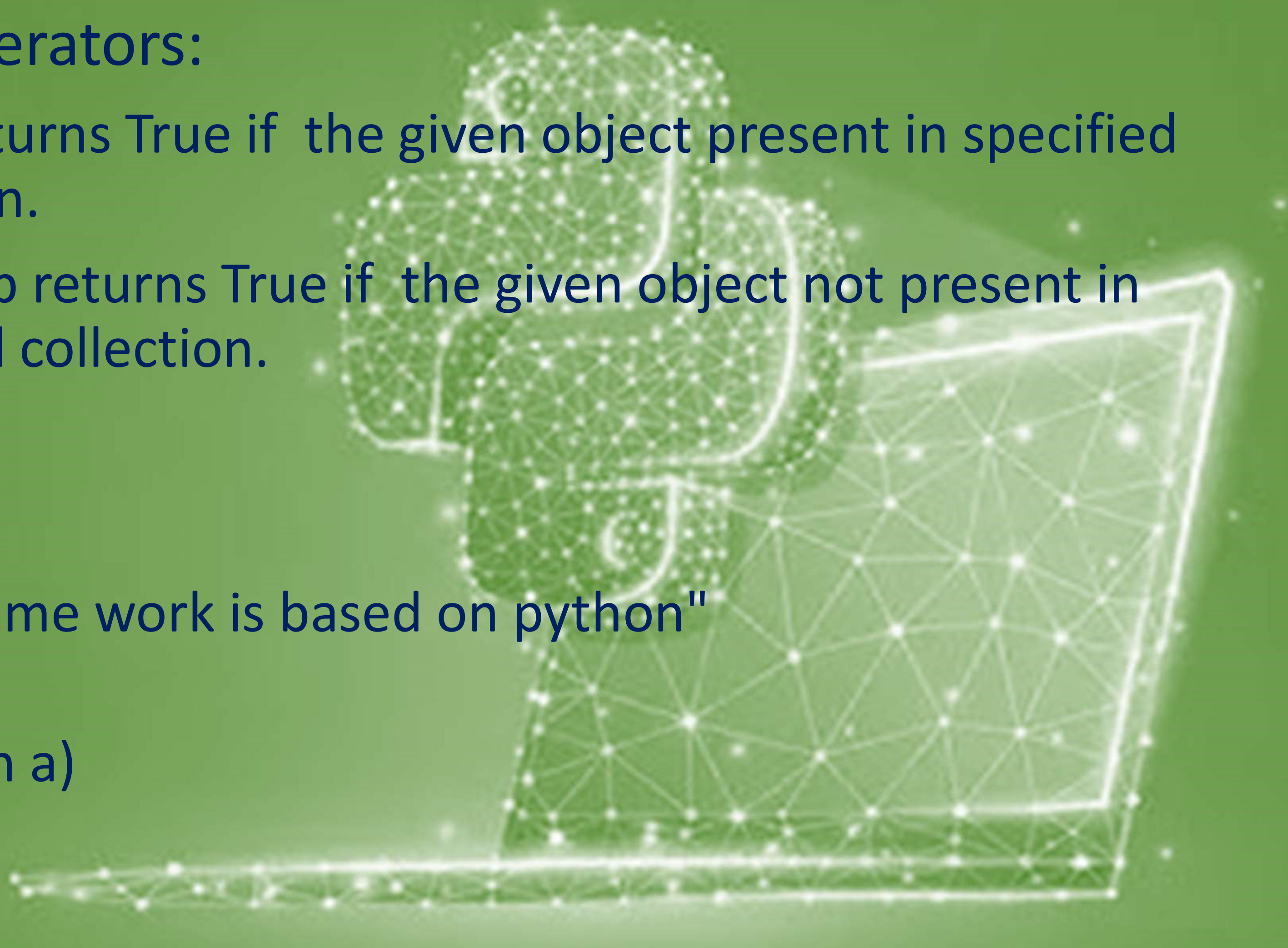
Example:

```
a="venu"
b="gopal"
c="venu"
print(a is not b)            #True
print(a is c)                 #True
print(b is c)                #False
print(c is not a)             #False
```

2. Membership operators:

i) in : a in b returns True if the given object present in specified collection.

ii)not in: a not in b returns True if the given object not present in specified collection.

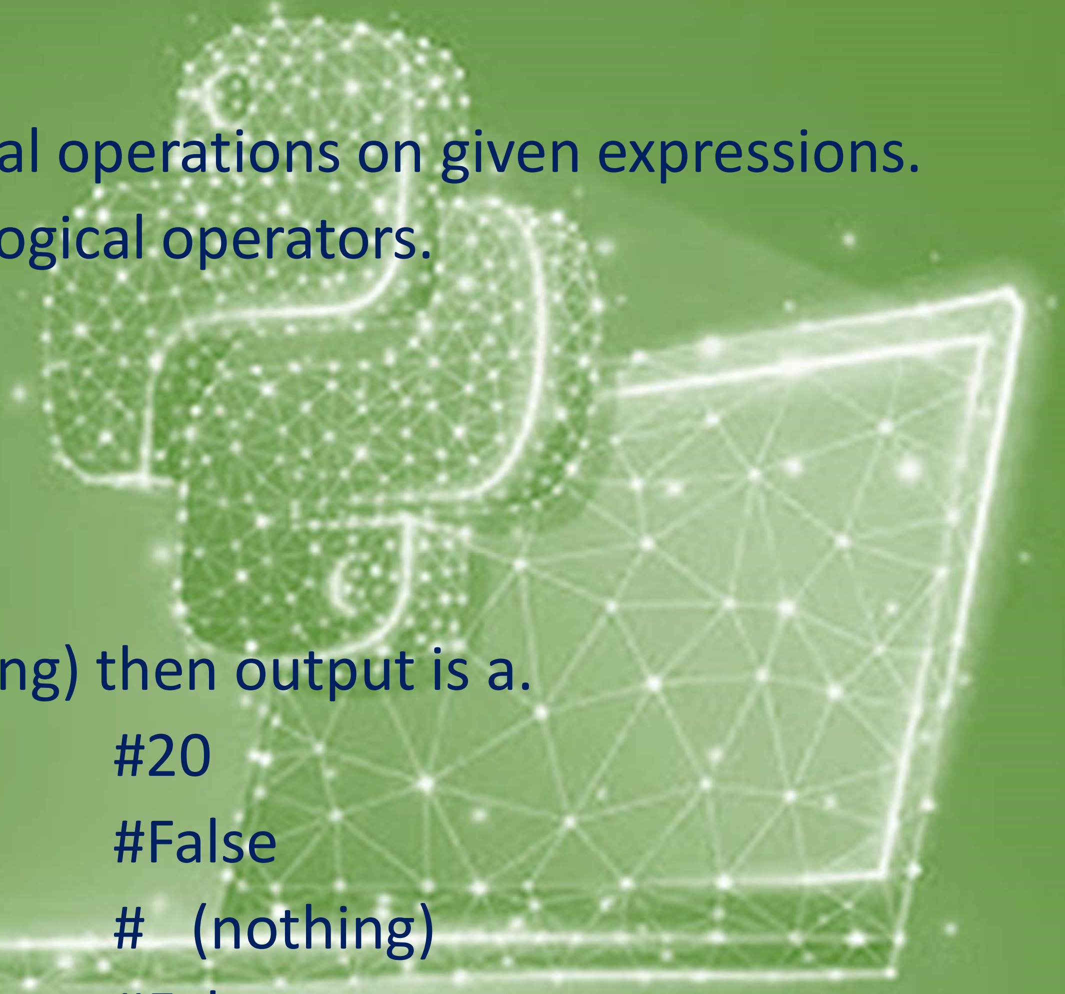


Example:

```
a="django web frame work is based on python"
print("web" in a)
print("india" not in a)
“””output:
    True
    True “””
```

Logical operators

- These are used to perform logical operations on given expressions.
- “and” ,” or” and “not” are the logical operators.



and:

Let print(a and b) is statement,

Case1: if a is True then output is b

Case2: is a is 0 or false or “”(nothing) then output is a.

```
Example: print(10 and 20)                    #20
         print(10 and False)                 #False
         print(“” and 20)                    #   (nothing)
         print( False and 20)                #False
```

or:

Let print(a or b) is statement,

Case1: if a is True then output is a

Case2: is a is 0 or false or “”(nothing) then output is b.

```
Example: print(10 or 20)           #10
         print(10 or False)        #10
         print(“” or 20)           # 20
         print( False or 20)       #20
```

not:

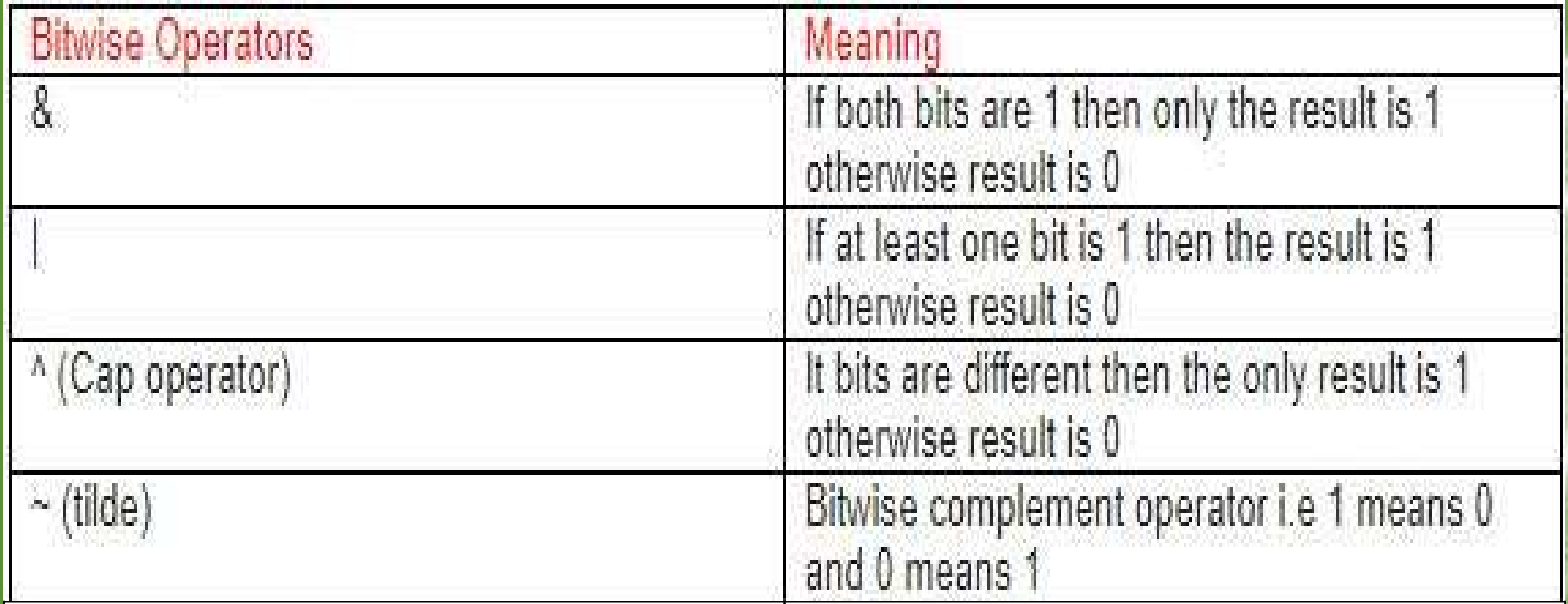
Let print(not a)

If a is True output is False.

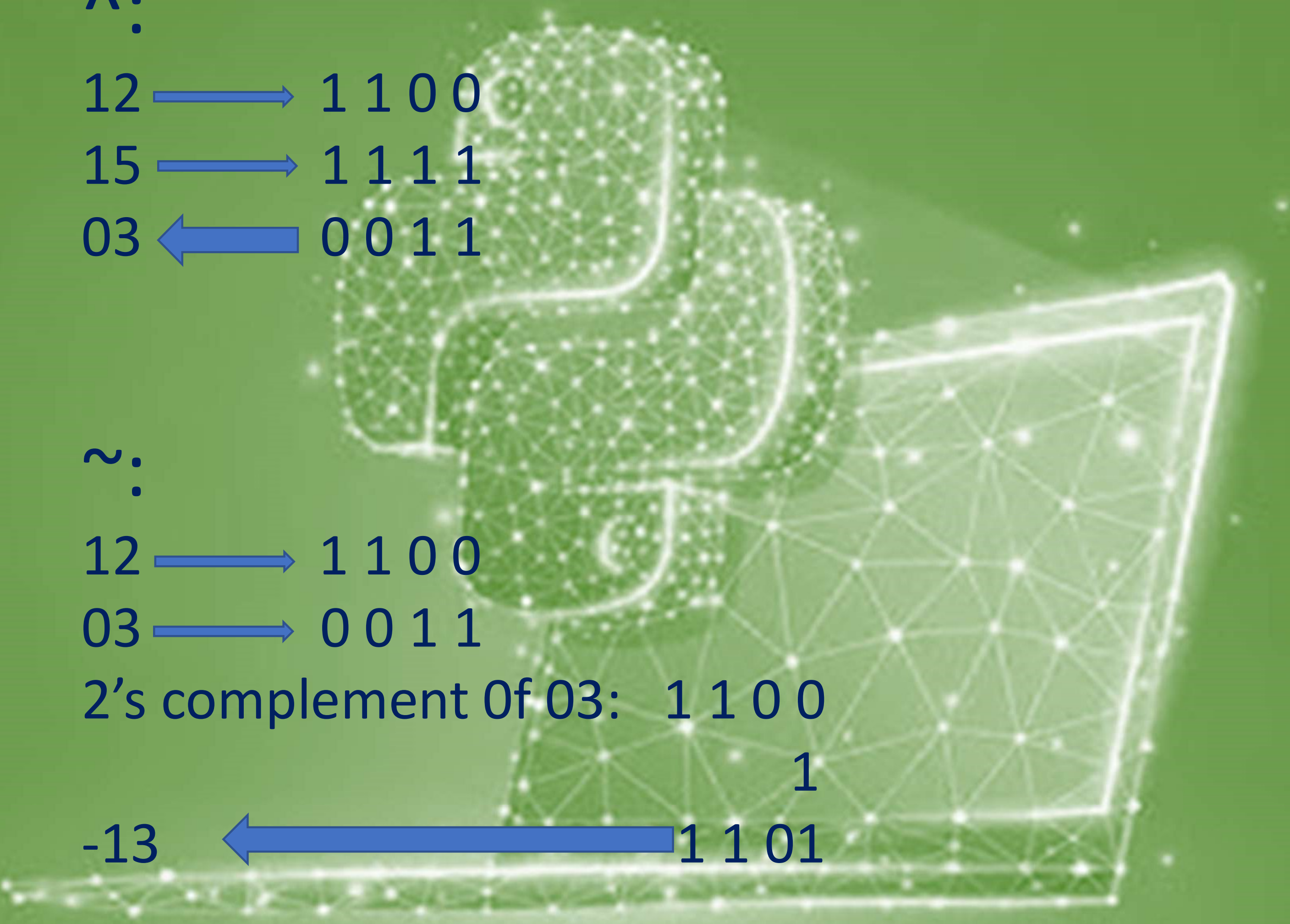
If a is False output is True.

Bitwise operators

Bitwise operations performs only on binary numbers



Bitwise Operators	Meaning
&	If both bits are 1 then only the result is 1 otherwise result is 0
|	If at least one bit is 1 then the result is 1 otherwise result is 0
^ (Cap operator)	It bits are different then the only result is 1 otherwise result is 0
~ (tilde)	Bitwise complement operator i.e 1 means 0 and 0 means 1
<<	Bitwise left shift operator
>>	Bitwise right shift operator



&:

12 → 1 1 0 0

15 → 1 1 1 1

12 ← 1 1 0 0

|:

12 → 1 1 0 0

15 → 1 1 1 1

15 ← 1 1 1 1

^:

12 → 1 1 0 0

15 → 1 1 1 1

03 ← 0 0 1 1

~:

12 → 1 1 0 0

03 → 0 0 1 1

2’s complement 0f 03: 1 1 0 0

1

-13 ← 1 1 01

Bitwise left shift operator(<<)

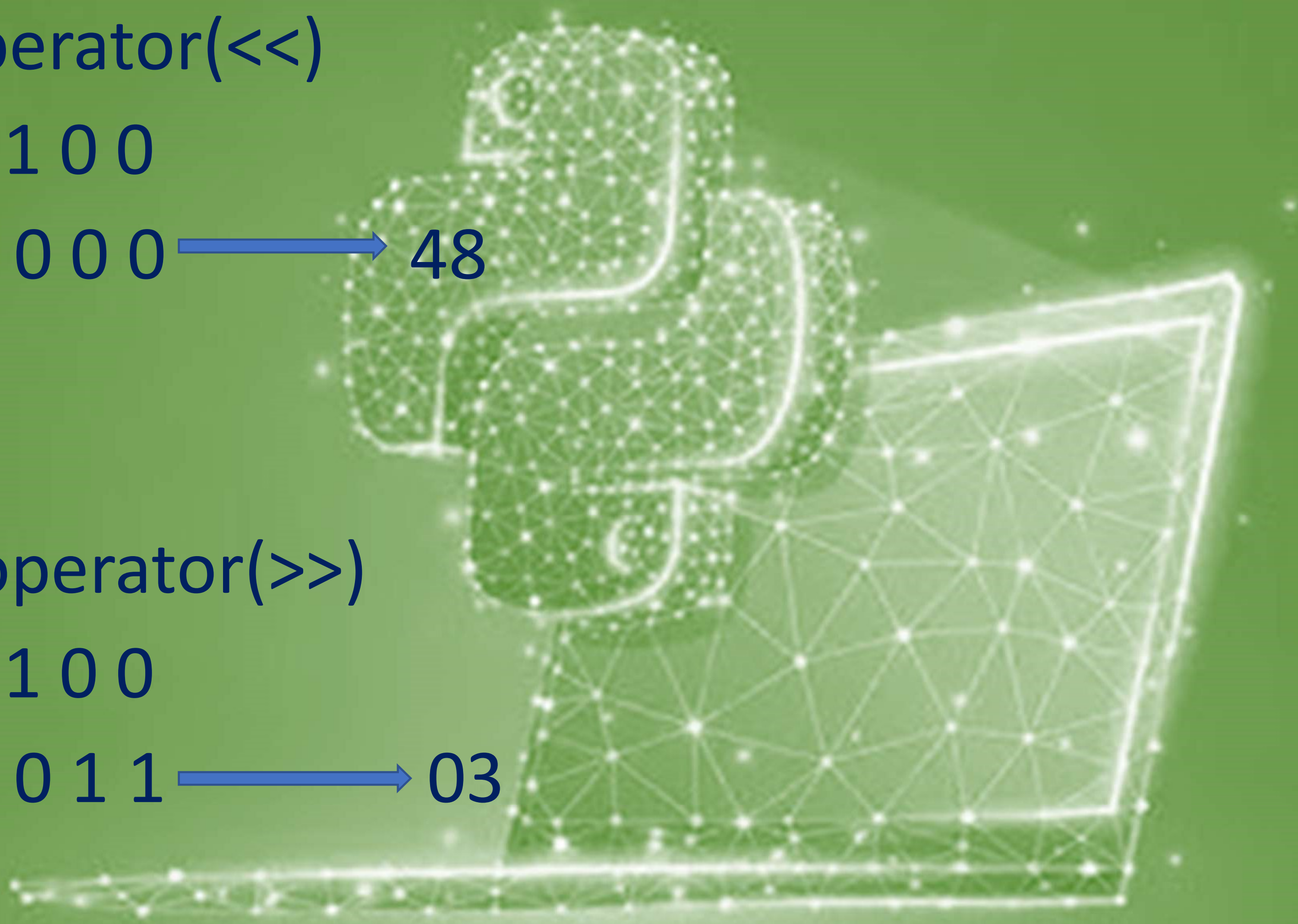
12 → 0 0 1 1 0 0

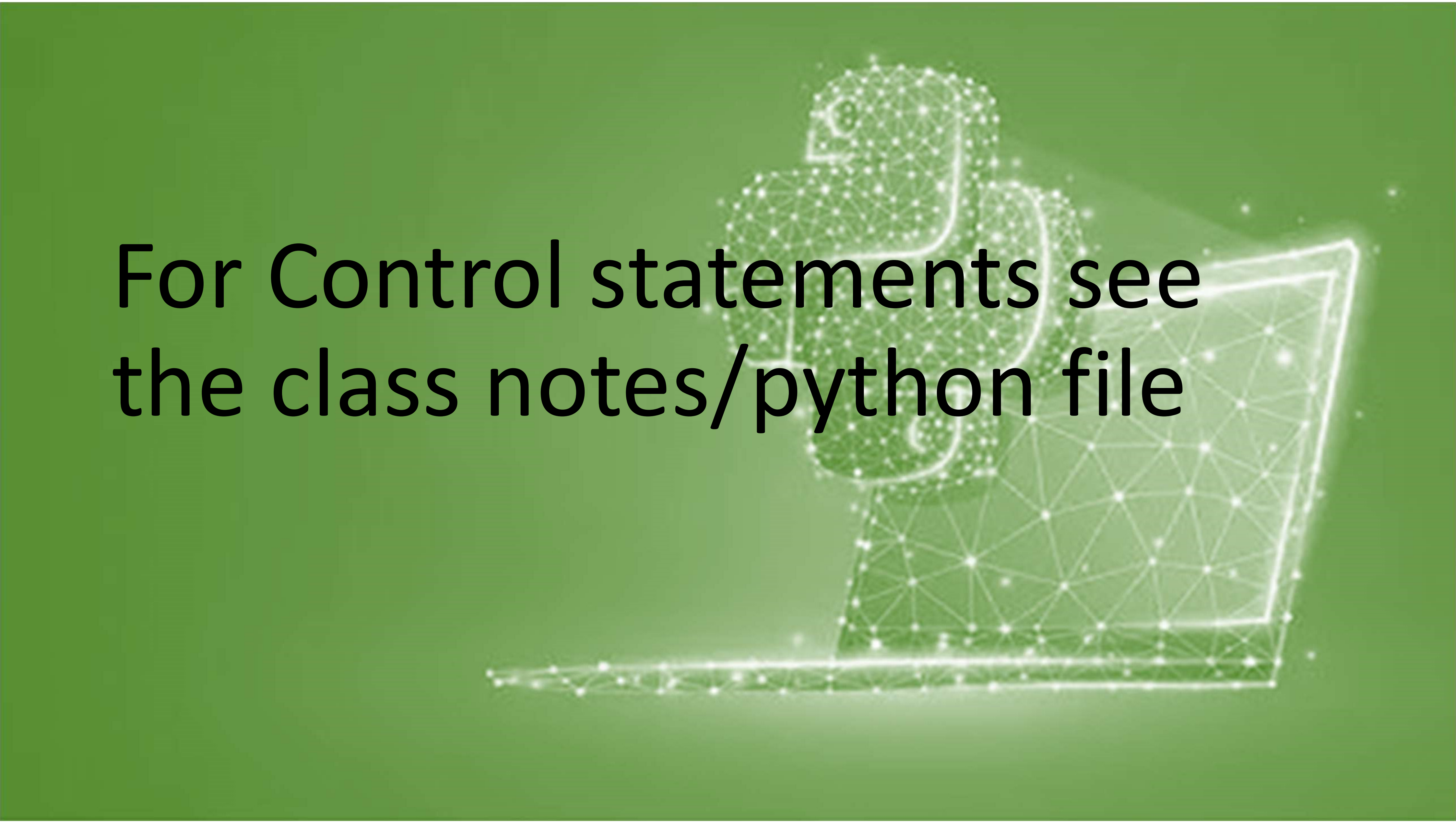
12<<2 → 1 1 0 0 0 0 → 48

Bitwise right shift operator(>>)

12 → 0 0 1 1 0 0

12>>2 → 0 0 0 0 1 1 → 03





For Control statements see the class notes/python file

Functions

➢Collection of statements is called function.

Advantages:

1) Repeated code will be avoided.
2) Re-usability of code.
3) Modularity
4) Debugging

Types of functions

1. Pre-defined functions: builtin python functions

Ex: print(),id(),type().....so on

2.User-defined functions: defined by users

Defining function:

```
 def fun_name(formal parameters):
         statement1
         statement2
         return x
```

Calling function

fun_name(actual parameters)

Ex:

```
 def div():
         a=b/c
         print(a)
div()
```

- Actual parameters (arguments) are optional
- Formal parameters are optional

Types of arguments

1. Required arguments:

```
def hello(a,b):
        print(a)
        print(b)
Hello(2,3)
```

2. Keyword arguments:

```
def hello(a,b):
        print(a)
        print(b)
hello(a=2,b=3)
```

3. Default arguments

```
        def hello(a=1,b=5):
                print(a)
                print(b)
        hello(2,3)
“””output:a=2 and b=3”””
```

4. Variable length arguments:

- i) Non-keyword arguments(*args)
- ii)keyword arguments(**args)

Non-keyword arguments(*args)

```
def hello(*a):
        print(a[0])
        print(a[1])
        print(type(a))
hello(2,3)
“””output:    2
              3
              <class   ‘tuple’>
```

keyword arguments(*args)

```
def student(**data):
        print(type(data))
        print(data)
        print(data.items())
        for i,j in data.items():
                print(f'{i} is {j}’)
student(name="venu", age=27)
```

```
“””output:
        <class 'dict’>
        {'name': 'venu', 'age': 27}
        dict_items([('name', 'venu'), ('age', 27)])
        name is venu
        age is 27
```

Local and Global Variable

- Local Variable:

  The variable which defines within a function i.e local to a function is called Local variable.

- Global Variable:

  The variable which defines outside the function i.e global to a the all functions is called Global variable.

Example:

```
a=5
print(a)
def f1():
    a=30
    g=globals()['a']
    print(a)
    print(g)
def f2():
    global a
    a=10
    print(a)
f1()
f2()
```

Output:

```
5
30
5
5
10
```

Recursion:

- A function calling itself is called Recursion.

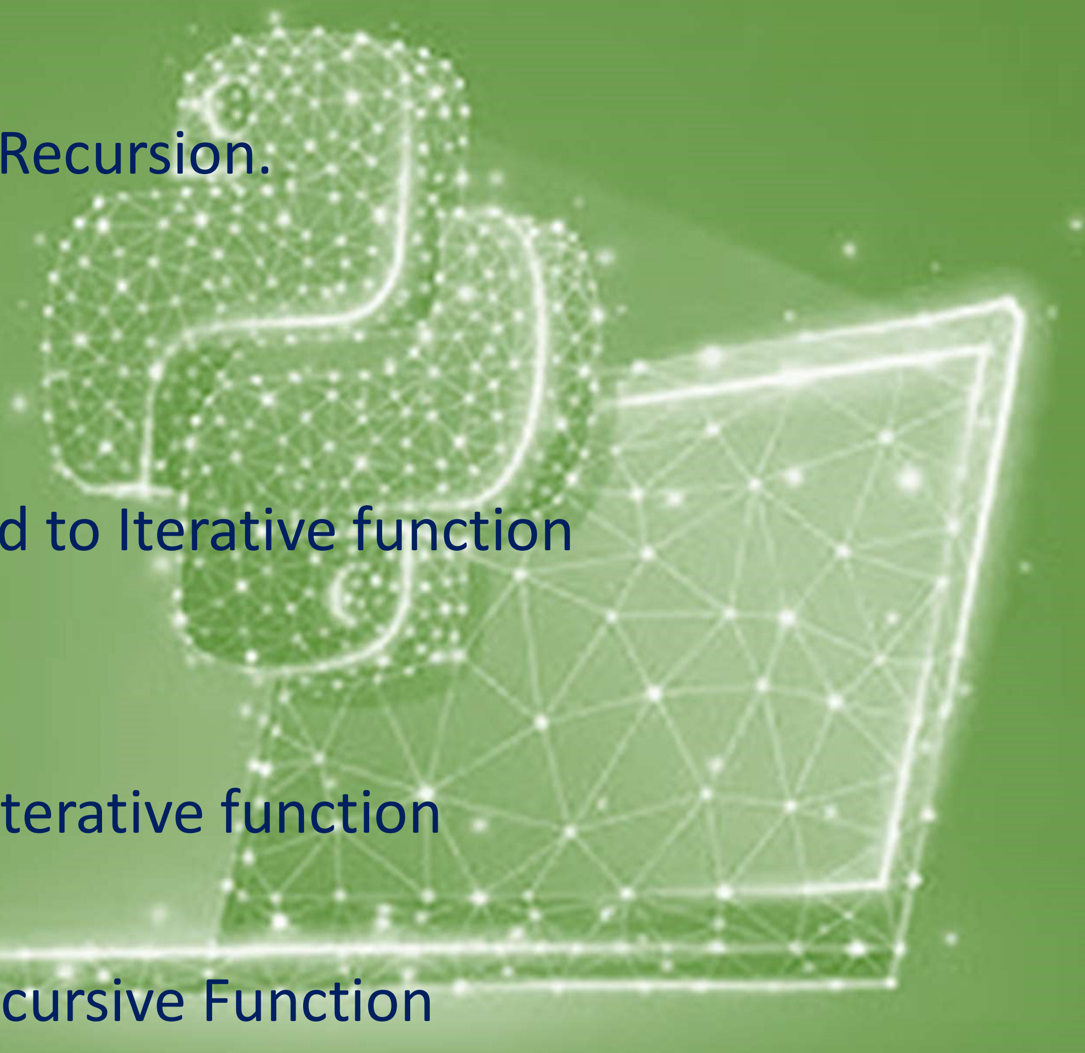
Advantages:

- Code is simple to understand
- Low memory occupies compared to Iterative function

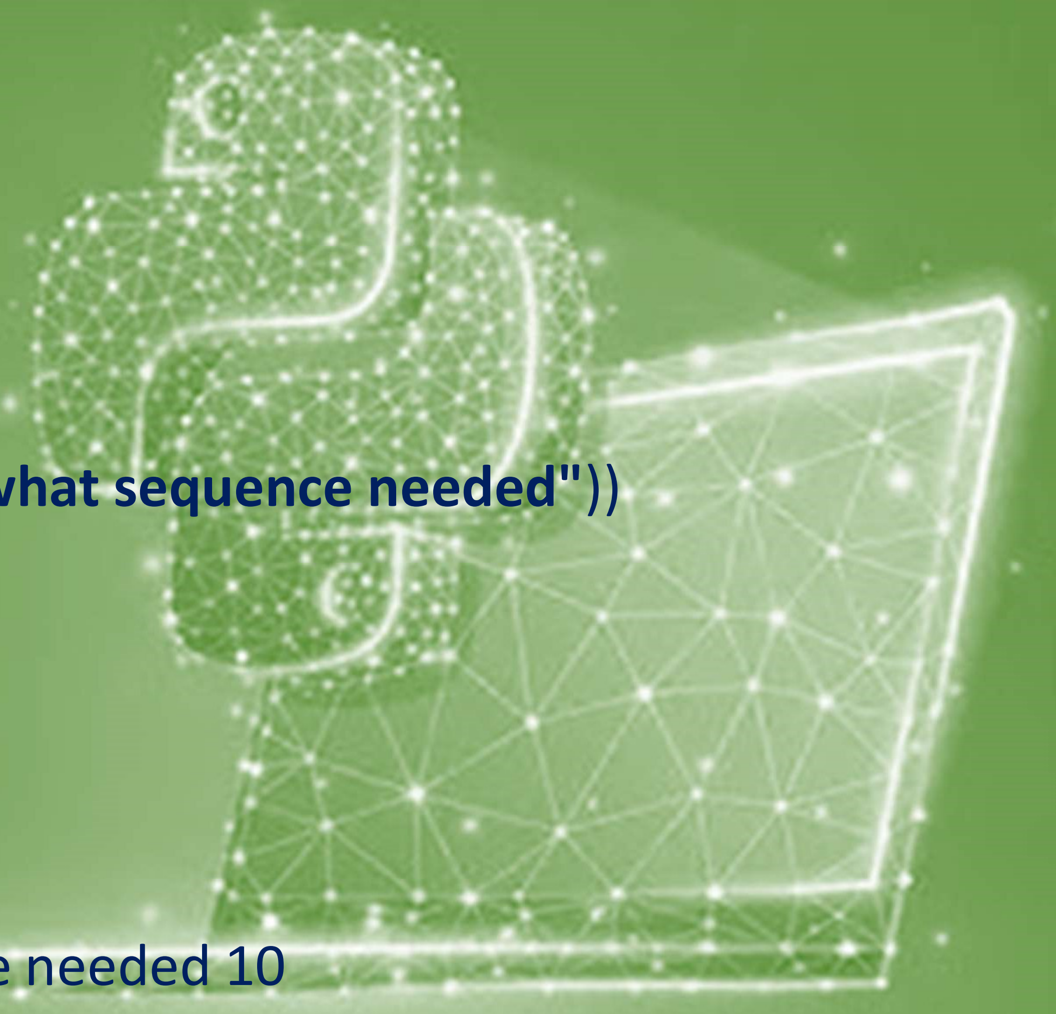
Disadvantages:

- Slow in execution compared to iterative function

Note: Stack concept Needed in Recursive Function



Example:



```
def incre(a,b,n):
   if a<=n:
      b=a+b
      return incre(a+1,b,n)
   return b
n=int(input("enter the number upto what sequence needed"))
a=0
b=0
l=incre(a,b,n)
print(l)
```

Output:

```
enter the number upto what sequence needed 10
55
```

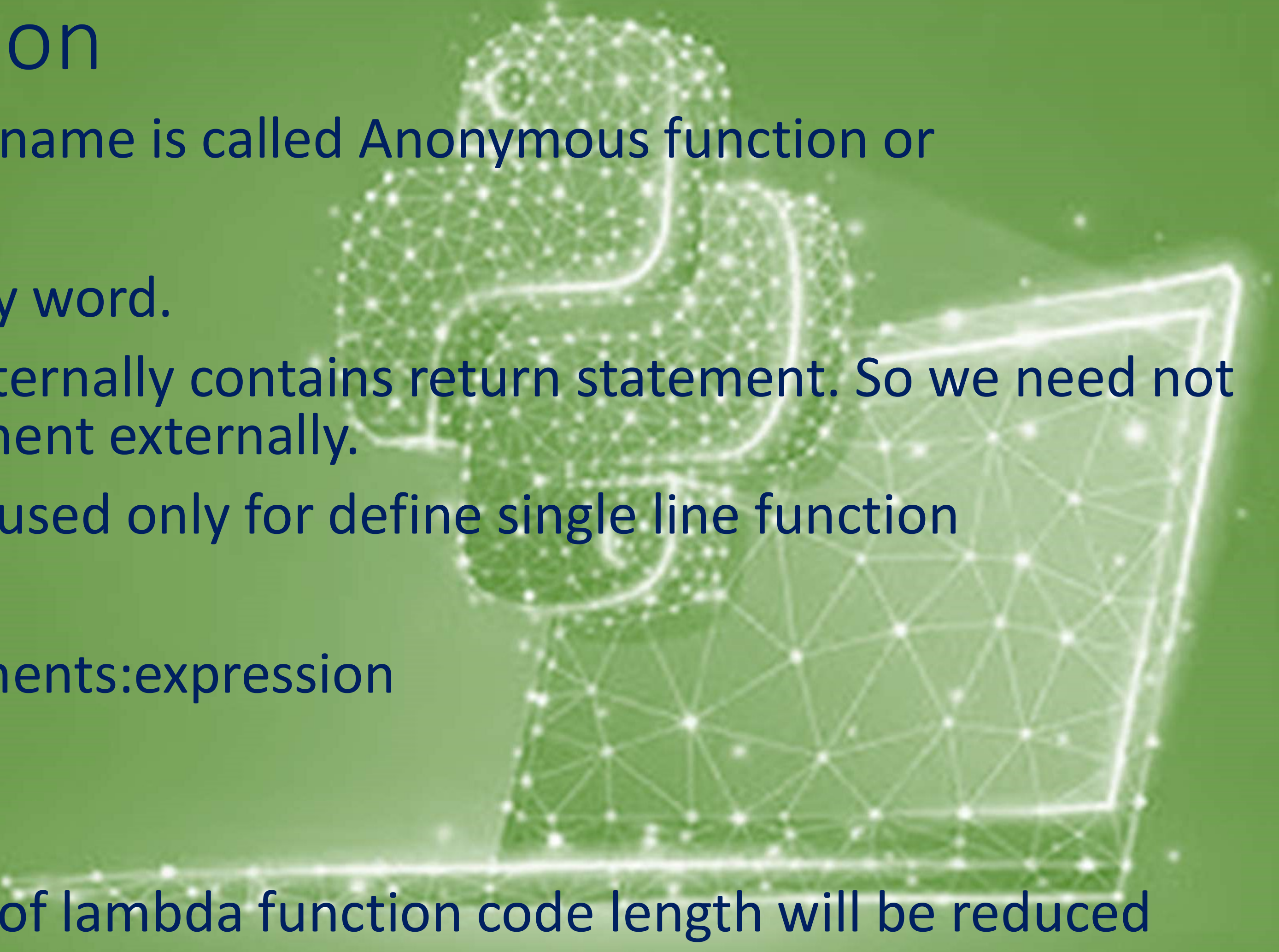
Lambda Function

- A function with no name is called Anonymous function or lambda function
- It uses “lambda” key word.
- Lambda function internally contains return statement. So we need not define return statement externally.
- Lambda function is used only for define single line function

Syntax: lambda arguments:expression

Ex: lambda n:n*n

Advantages: Because of lambda function code length will be reduced



Example:

Without lambda:

```
def quadratic_expression(a,b,c,x):
   return a*x**2+b*x+c
a=quadratic_expression(1,2,3,2)
print(a)
```

With lambda:

```
n=lambda a,b,c,x:a*x**2+b*x+c
print(n(1,2,3,2))
“””output is
11”””
```

filter():

- ➢It filters the values or elements based on given condition

Syntax: filter(function,iterable/sequence)

Ex:

```
even=lambda h:h%2!=0
a=[83,43,56,633,5,22,2,3,4,55]
k=list(filter(even,a))
print(k)
```

Output: [83,43,633,5,3,55]

map():

➢It makes operations on elements of a sequence or iterables

Note: After operations length os iterables or sequence remains same

Syntax: map(function, iterable/sequence)

EX:

```
even=lambda h:h*2
a=[8,4,5,33,22,3,55]
k=list(map(even,a))
print(k)
“””output : [16,8,10,66,44,9,110]”””
```

reduce():

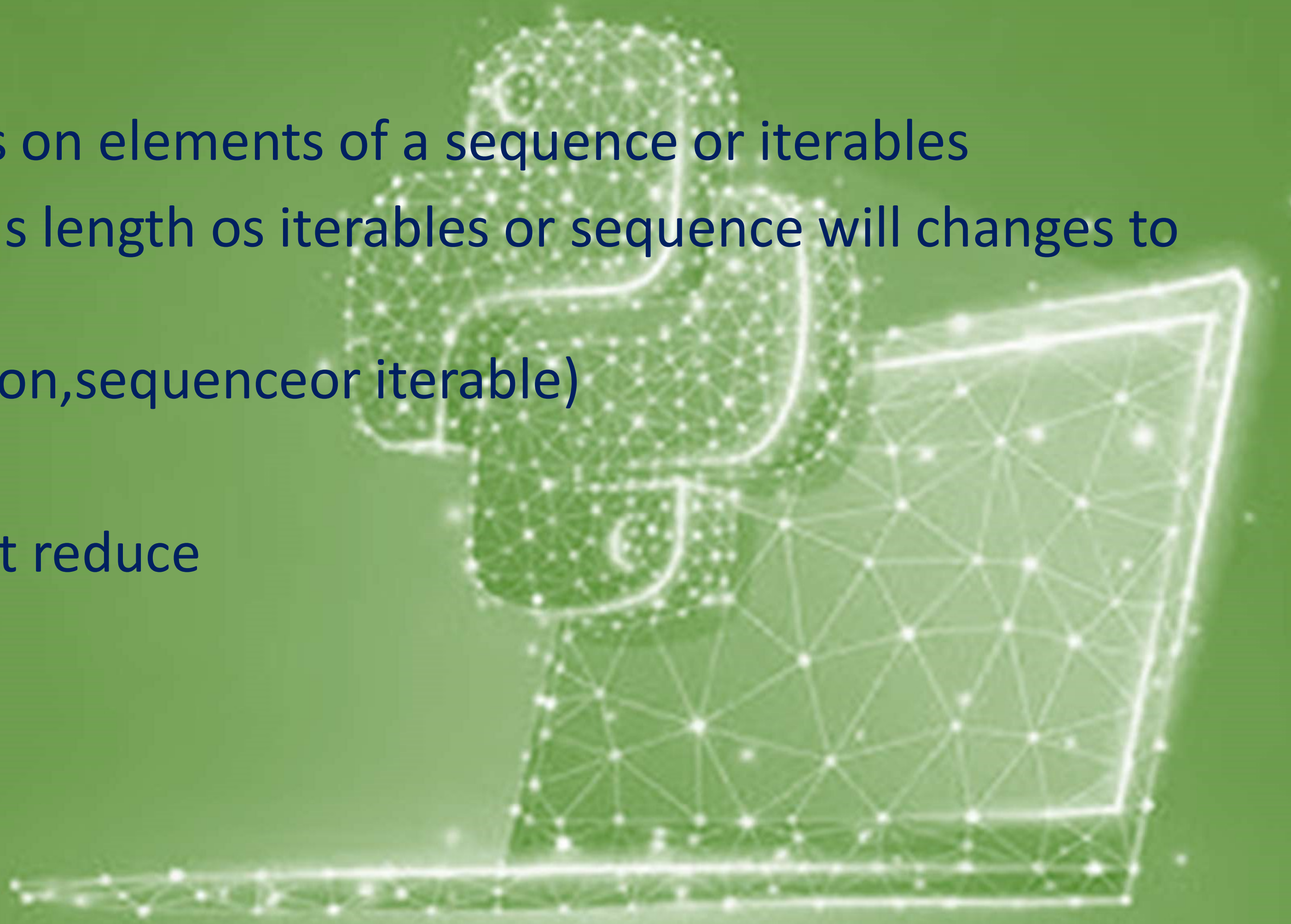
- It makes operations on elements of a sequence or iterables

Note: After operations length os iterables or sequence will changes to single value

Syntax: reduce(function,sequenceor iterable)

Ex:

```
from functools import reduce
even=lambda h,l:h+l
a=[43,56,6,5,3,4,55]
k=reduce(even,a)
print(k)
“””Output: 172”””
```

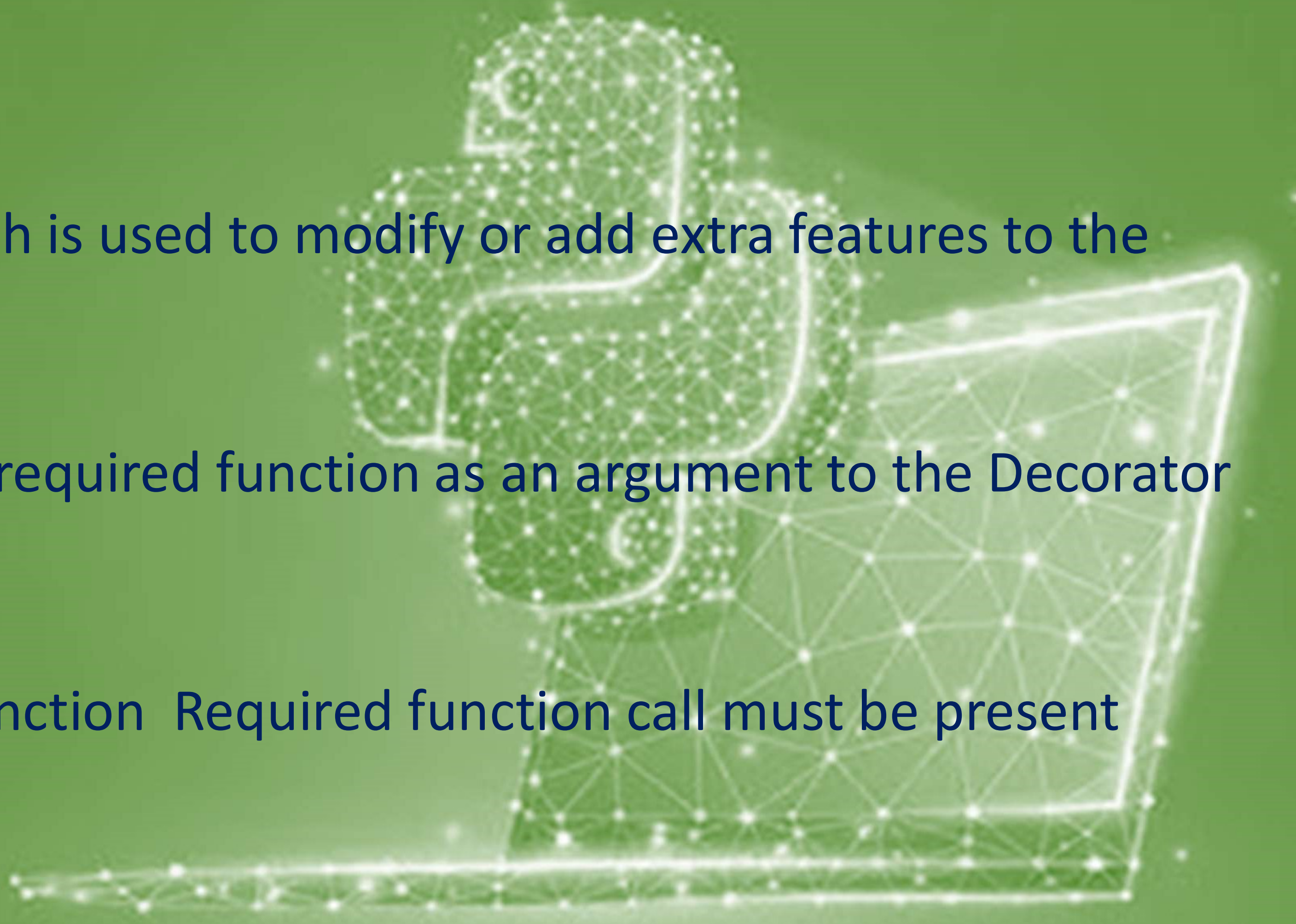


Decorators

- It is a function which is used to modify or add extra features to the existing function.

- In this we pass the required function as an argument to the Decorator function.

Note: In decorator function Required function call must be present



Example:

```
def smart_div(fun):
   def inner(x,y):
      if x<y:
         x,y=y,x
      return fun(x,y)
   return inner

@smart_div
def div(a,b):
   print(a/b)
div(2,4)
```

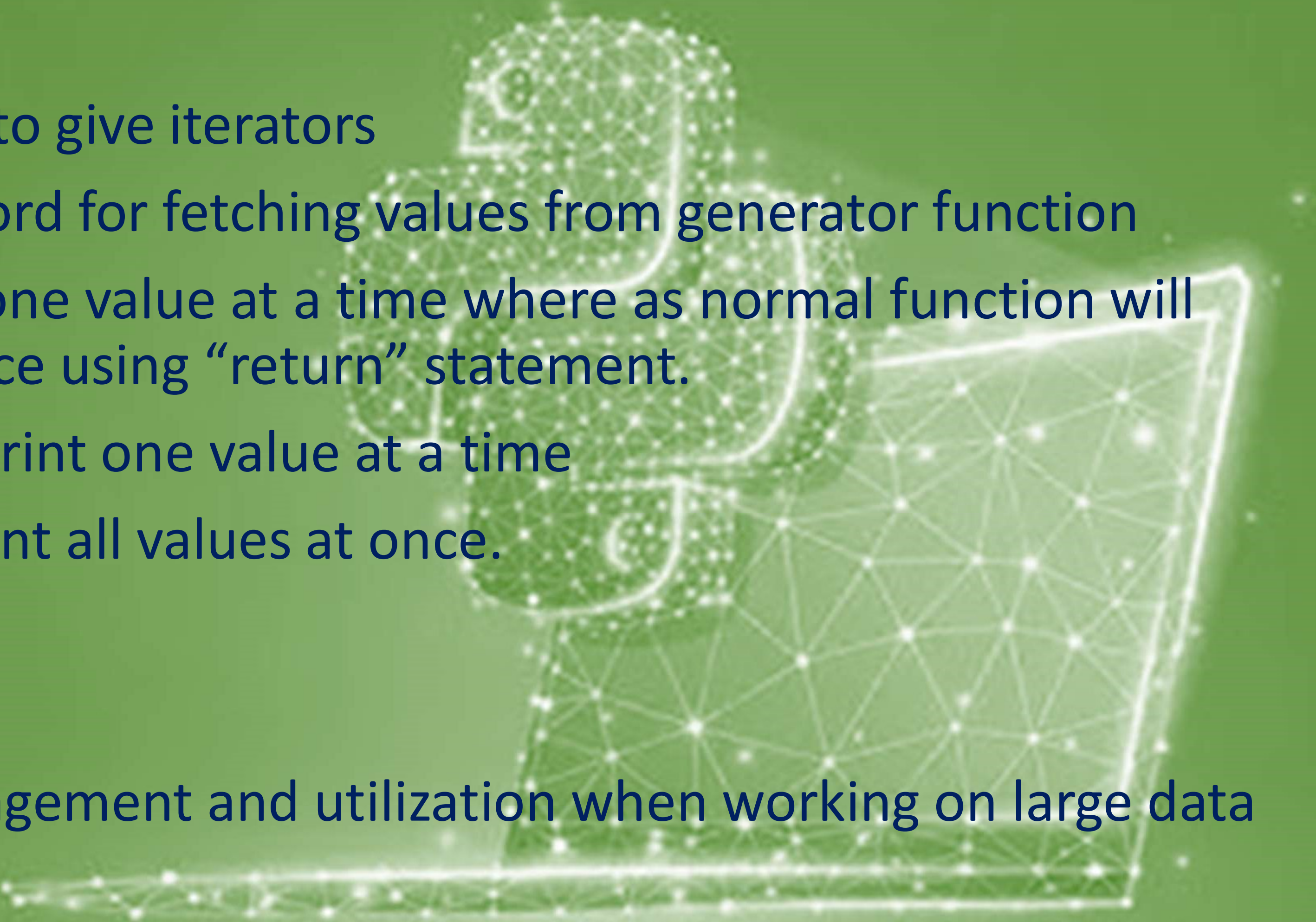
```
“””output:
     2.0
“””
```

Generators:

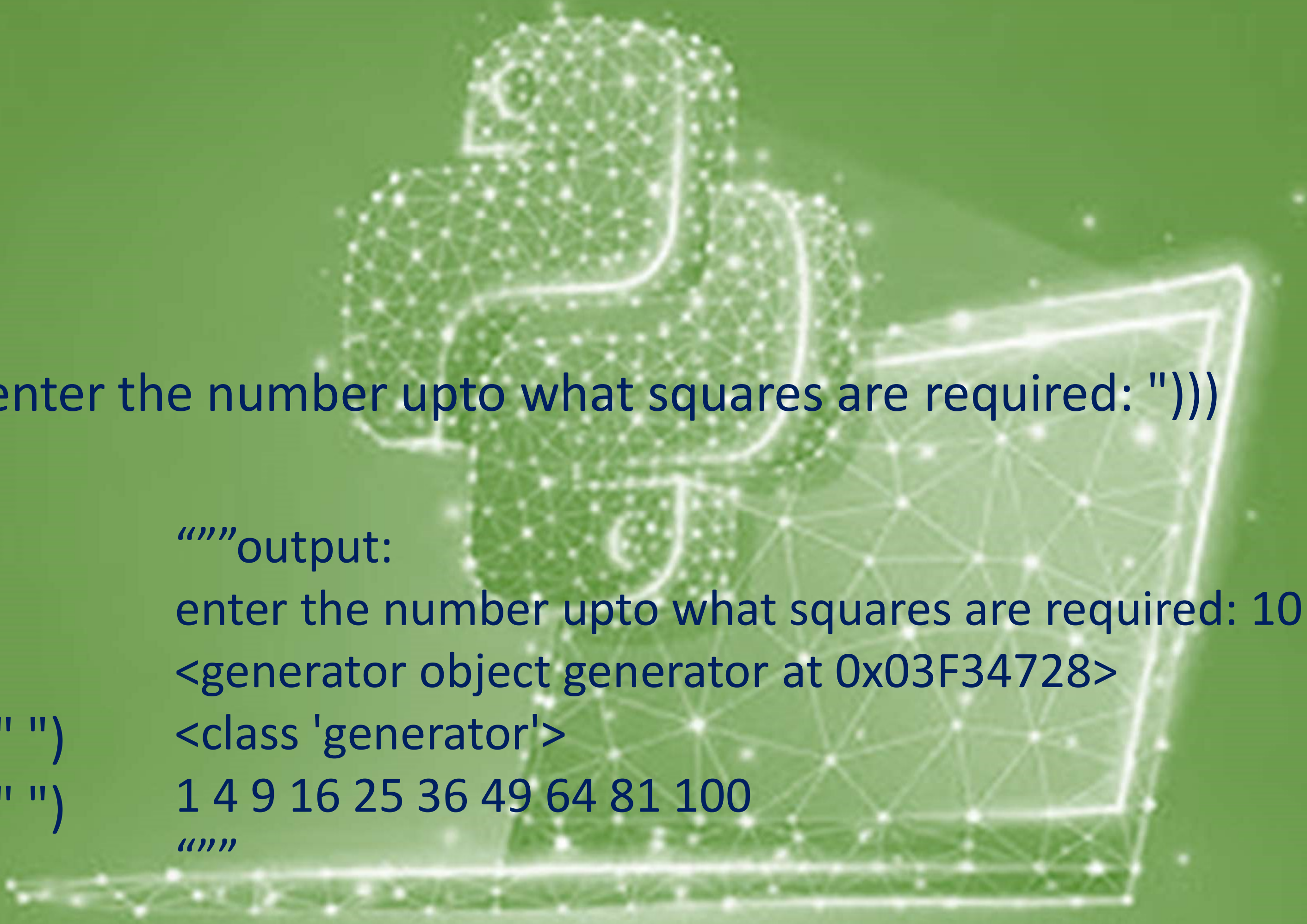
- Generators are used to give iterators
- We use “yield” keyword for fetching values from generator function
- Generators will give one value at a time where as normal function will give entire data at once using “return” statement.
- next() is used for to print one value at a time
- for loop is used to print all values at once.

Advantages:

- Better memory management and utilization when working on large data sets.
- Can produce infinite items



Example:



```
def generator(b):
    for i in range(b):
        i+=1
        yield i*i
a=generator(int(input("enter the number upto what squares are required: ")))
print(a)
print(type(a))
print(next(a),end=" ")
print(next(a),end=" ")
print(a.__next__(),end=" ")
print(a.__next__(),end=" ")
for i in a:
    print(i,end=" ")
```

```
“””output:
enter the number upto what squares are required: 10
<generator object generator at 0x03F34728>
<class 'generator'>
1 4 9 16 25 36 49 64 81 100
“””
```