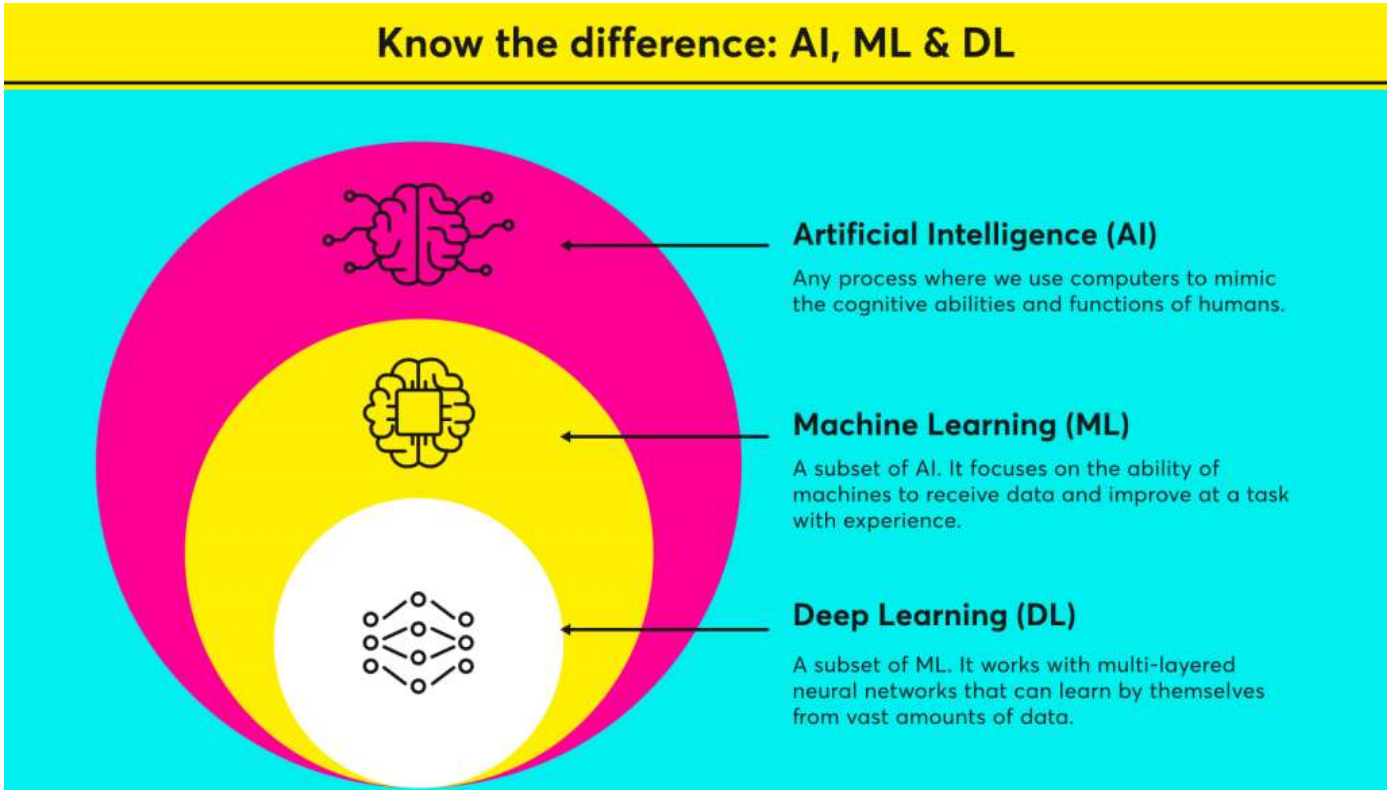
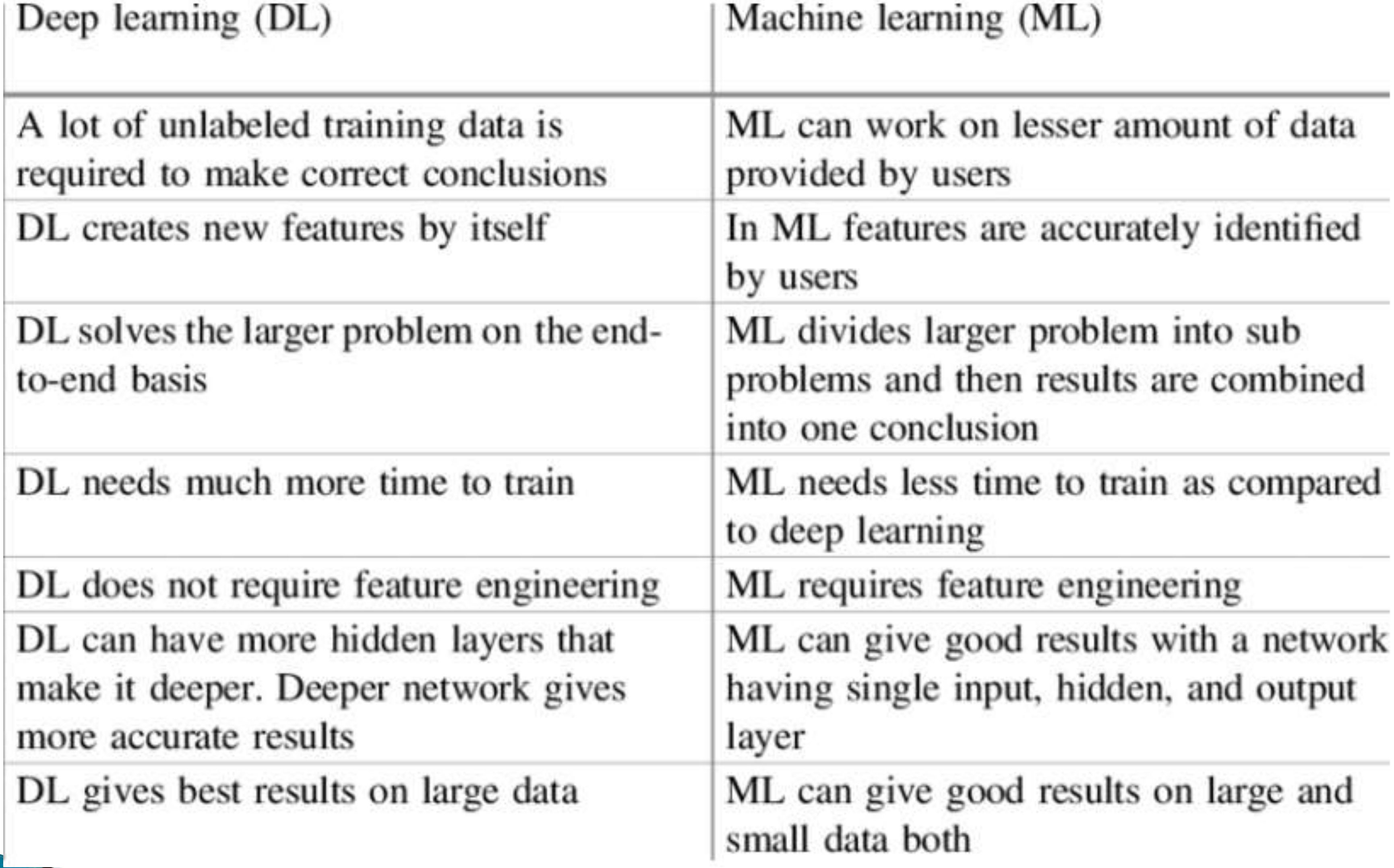
ARTIFICIAL INTELLIGENCE CLASS-4☺

BY
GIRISH
KEERTHIVASAN

Difference Between ML & DL


Know the difference: AI, ML & DL
Artificial Intelligence (AI)
Any process where we use computers to mimic the cognitive abilities and functions of humans.
Machine Learning (ML)
A subset of AI. It focuses on the ability of machines to receive data and improve at a task with experience.
Deep Learning (DL)
A subset of ML. It works with multi-layered neural networks that can learn by themselves from vast amounts of data.



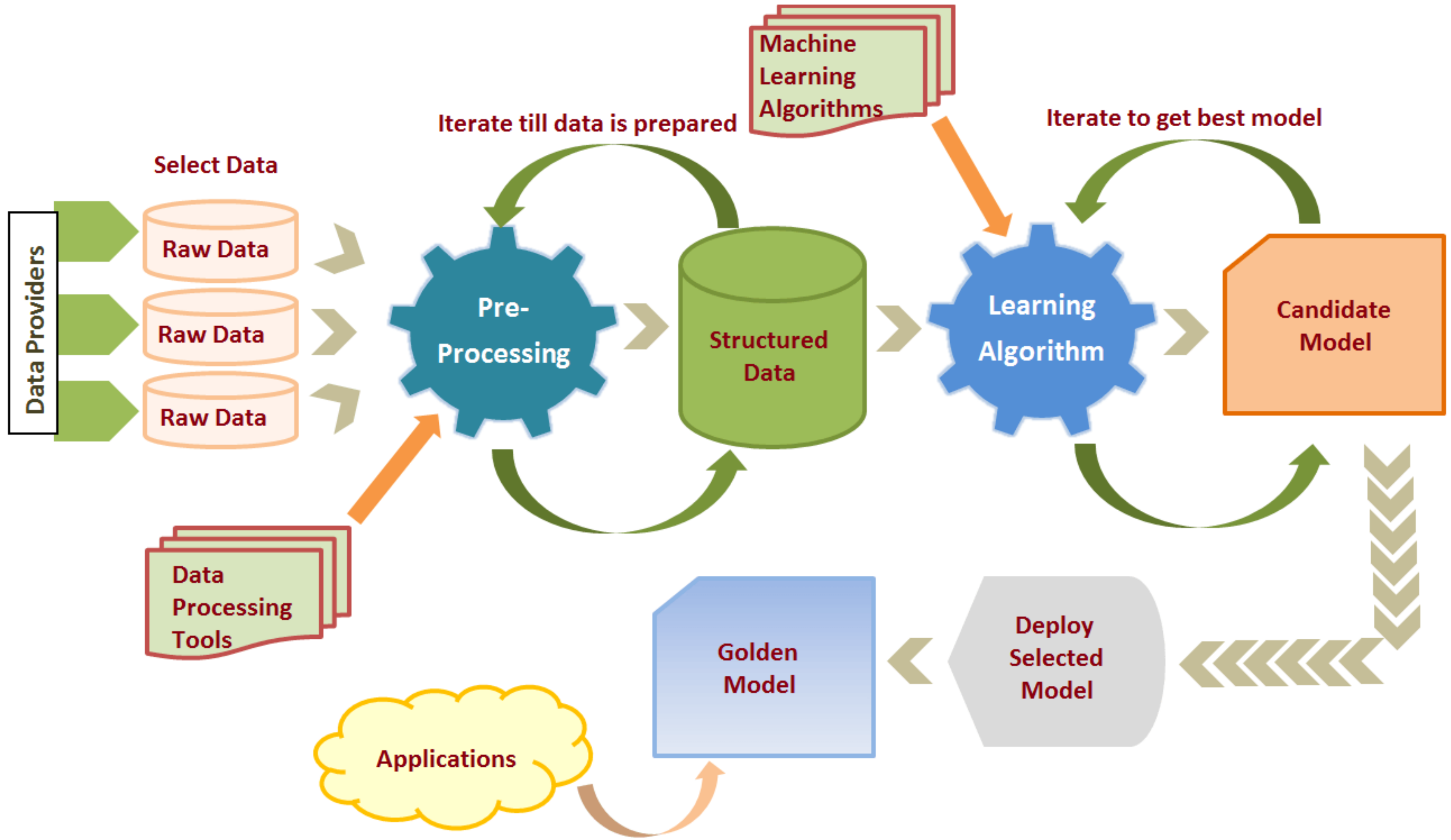
Deep learning (DL)	Machine learning (ML)
A lot of unlabeled training data is required to make correct conclusions	ML can work on lesser amount of data provided by users
DL creates new features by itself	In ML features are accurately identified by users
DL solves the larger problem on the end-to-end basis	ML divides larger problem into sub problems and then results are combined into one conclusion
DL needs much more time to train	ML needs less time to train as compared to deep learning
DL does not require feature engineering	ML requires feature engineering
DL can have more hidden layers that make it deeper. Deeper network gives more accurate results	ML can give good results with a network having single input, hidden, and output layer
DL gives best results on large data	ML can give good results on large and small data both

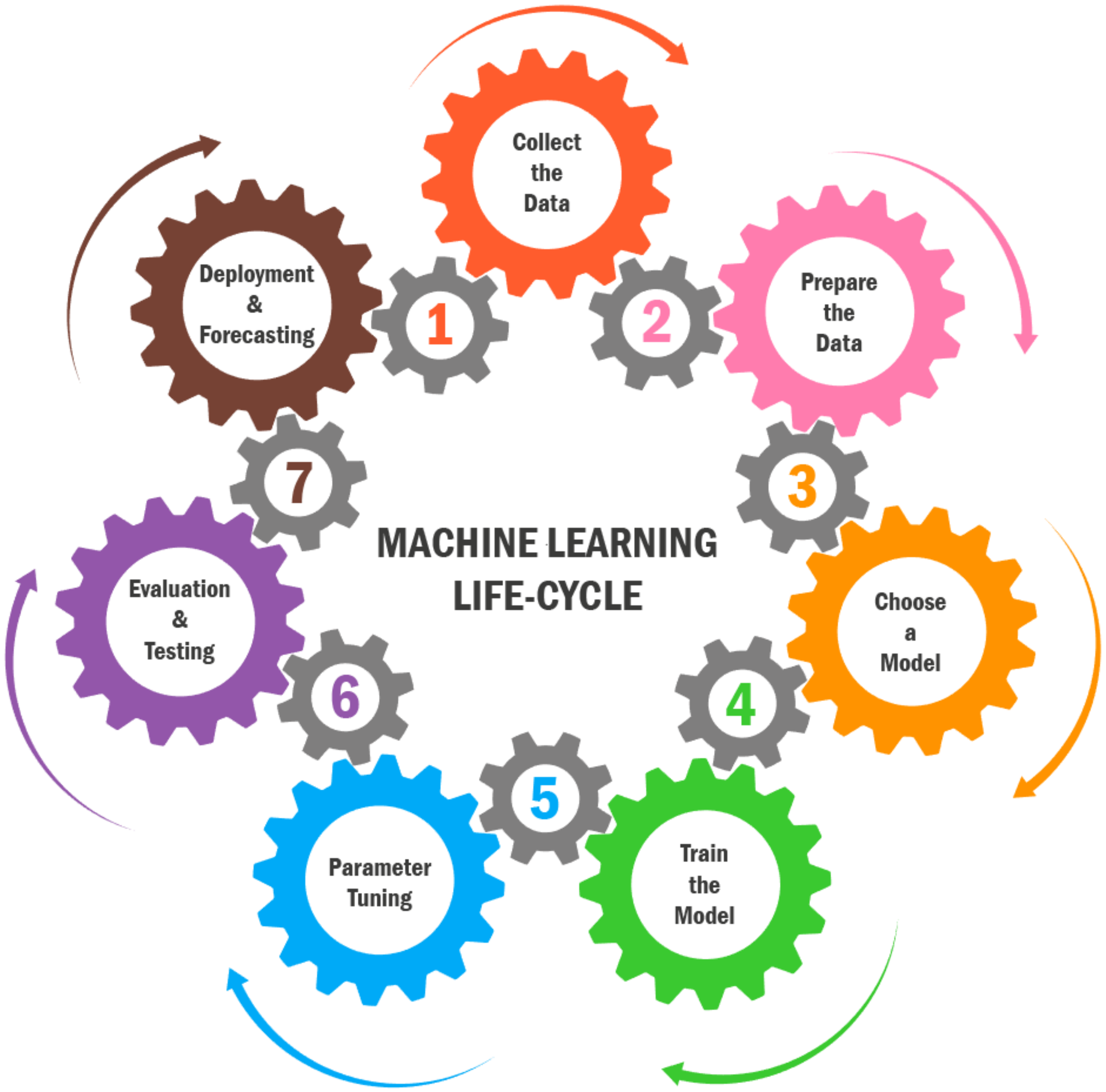
Why Deep Learning is necessary?

- Deep learning eliminates some of data pre-processing that is typically involved with machine learning.

- These algorithms can ingest and process unstructured data, like text and images, and it automates feature extraction, removing some of the dependency on human experts.

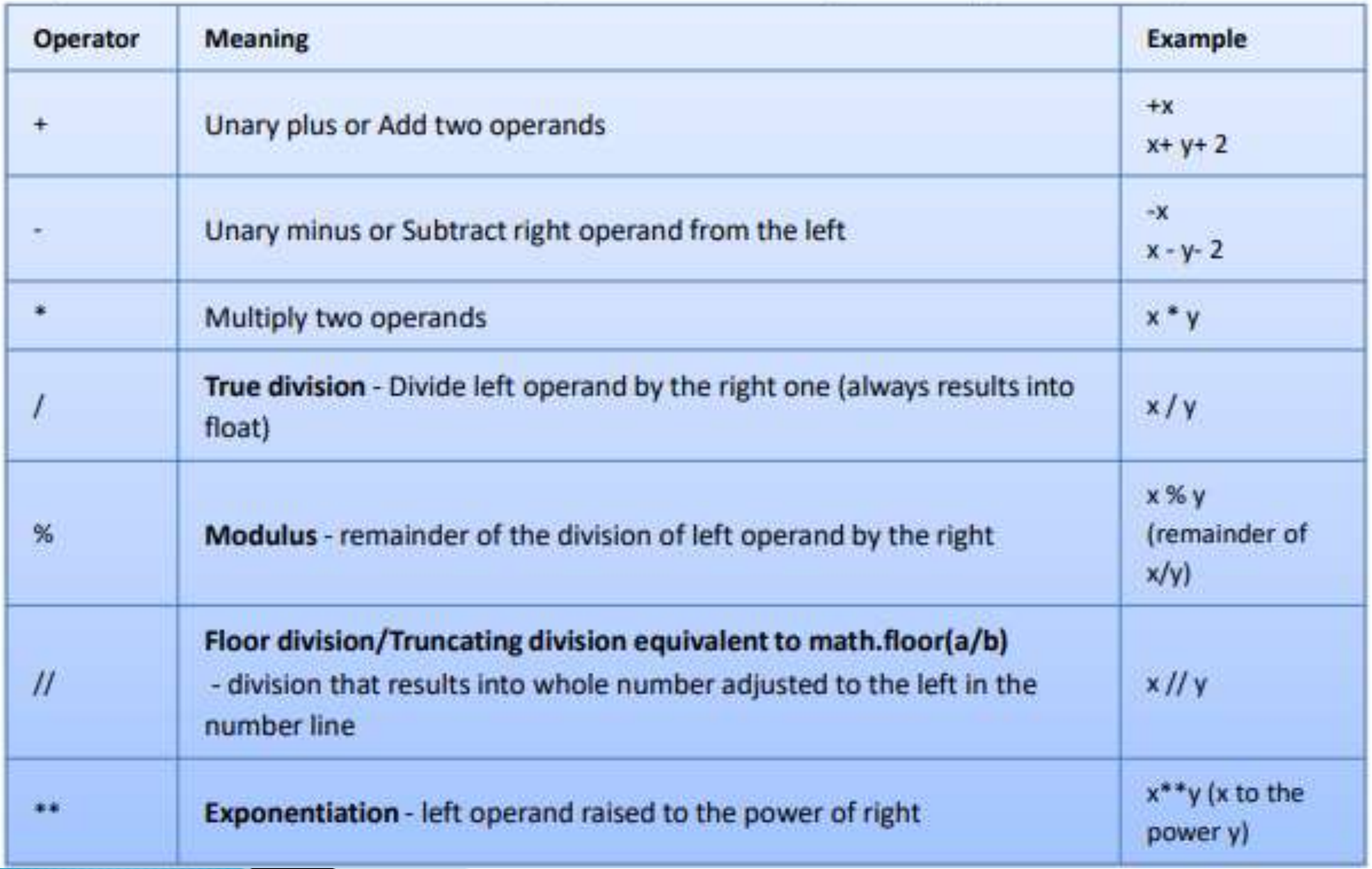
Steps in Machine Learning Process Flow View(Understanding)




MACHINE LEARNING LIFE-CYCLE
1
Collect the Data
2
Prepare the Data
3
Choose a Model
4
Train the Model
5
Parameter Tuning
6
Evaluation & Testing
7
Deployment & Forecasting

Arithmetic Operators

- Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, etc.



Operator	Meaning	Example
+	Unary plus or Add two operands	+x x+ y+ 2
-	Unary minus or Subtract right operand from the left	-x x - y- 2
*	Multiply two operands	x * y
/	**True division** - Divide left operand by the right one (always results into float)	x / y
%	**Modulus** - remainder of the division of left operand by the right	x % y (remainder of x/y)
//	**Floor division/Truncating division equivalent to math.floor(a/b)** - division that results into whole number adjusted to the left in the number line	x // y
**	**Exponentiation** - left operand raised to the power of right	x**y (x to the power y)

Arithmetic Operators Cont'd

Arithmetic Operators	Example	Result
+ (Addition)	10+25	35
- (Negation, Subtraction)	-10 10-25	-10 -15
*(Multiplication)	10*5	50
/(Division)	25/10	2.5
//(Truncated Division)	25//10 25//10.0	2 2.0
%(Modulus)	25%10	5
**(Exponentiation)	10**2	100

CONDITIONAL STATEMENTS

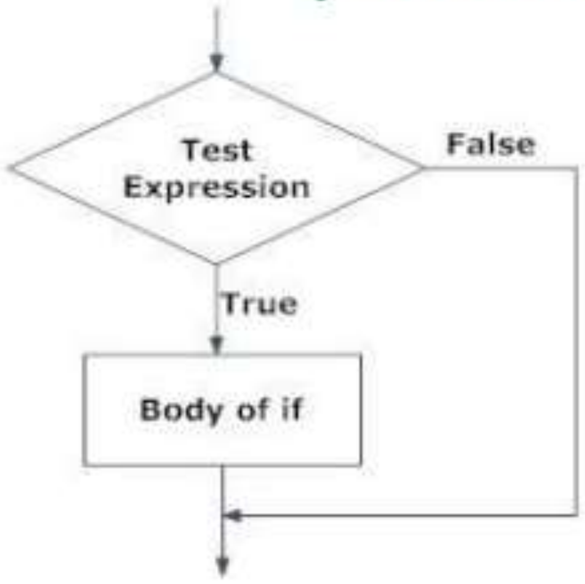
- if statement
- if…else statement

LOOP STRUCTURES

- for
- while

If statement

(Checks only one condition)



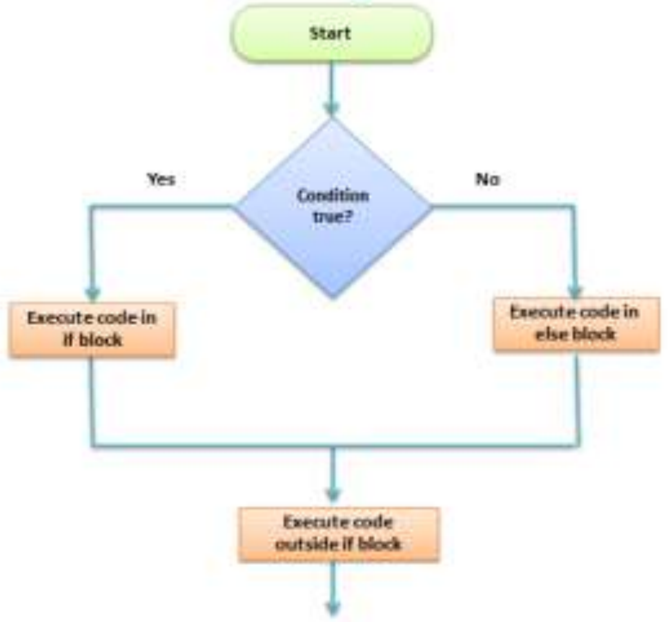
- Here, the **program evaluates the test expression and will execute statement(s) only if the test expression is True.**
- If the test expression is False, the statement(s) is not executed.
- In Python, the body of the if statement is indicated by the indentation.
- **The body starts with an indentation** and the first unindented line marks the end.
- Python interprets non-zero values as True.

<u>if Statement Syntax</u>

```
if test expression:
    statement(s)
```

if...else statement

(To check two conditions)



- The if..else statement evaluates test expression and will execute the body of if only when the test condition is True.
- If the condition is False, the body of else is executed.
- Indentation is used to separate the blocks.
- A **header** in Python is a specific keyword followed by a colon.
- The set of statements following a header in Python is called a **suite** (commonly called a **block**)
- A header and its associated suite are together referred to as a **clause**.
- A **compound statement** in Python may consist of one or more clauses.

Syntax of if...else [compound statement]

$$\left.\begin{array}{l}\text{if test expression:} \rightarrow \textbf{header} \\ \quad \text{Body of if} \longrightarrow \text{suite}\end{array}\right\} \textbf{Clause}$$

$$\left.\begin{array}{l}\text{else:} \longrightarrow \textbf{header} \\ \quad \text{Body of else} \longrightarrow \text{suite}\end{array}\right\} \textbf{Clause}$$

if…else statement Cont’d

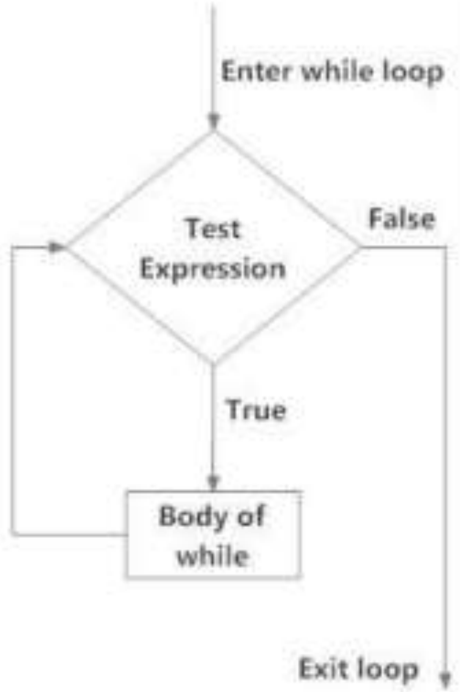
#Write a program to check whether given number is +ve or -Ve

```
num = int(input("Enter num:"))
if num < 0:
    print(num, "is a negative number.")
else:
    print(num, "is a positive number.")
```

Output:

```
Enter num:12
12 is a positive number.

Enter num:-15
-15 is a negative number.
```

While loop



- This loop is used to **execute multiple statement or codes repeatedly until the given condition is true**
- Mostly while loop is used when we don't know the number of times to iterate (indefinite loops)

Syntax:

```
while <expr>:
    <statement(s)>
```

- <statement(s)> represents the block to be repeatedly executed /(or)body of the loop.
- This is denoted with indentation
- First unintended line marks the end
- The controlling <expr>, typically involves one or more variables that are initialized prior to starting the loop and then modified somewhere in the body of the loop

While loop Example

```
n=5
while n>0:
    print("n=",n)
    n=n-1
```

Output:

```
n= 5
n= 4
n= 3
n= 2
n= 1
>>>
```

While loop working

Python Code	**Working**

```
n=5
while n>0:
    print("n=",n)
    n=n-1
```
	n=5

output

While loop working

Python Code

```
n=5
while n>0:
    print("n=",n)
    n=n-1
```

Working

n=5
5>0
True

output

While loop working

Python Code	Working

`n=5`
`while n>0:`
`    print("n=",n)`
`    n=n-1`
	n=5
	5>0
	True

output

```
n=5
```

While loop working

Python Code

```
n=5
while n>0:
    print("n=",n)
    n=n-1
```

Working

n=4

output

```
n=5
```

While loop working

Python Code	**Working**

`n=5`
`while n>0:`
`    print("n=",n)`
`    n=n-1`
	n=4
	4>0
	True

output

```
n=5
```

While loop working

Python Code

```
n=5
while n>0:
    print("n=",n)
    n=n-1
```

Working

n=4

4>0

True

output

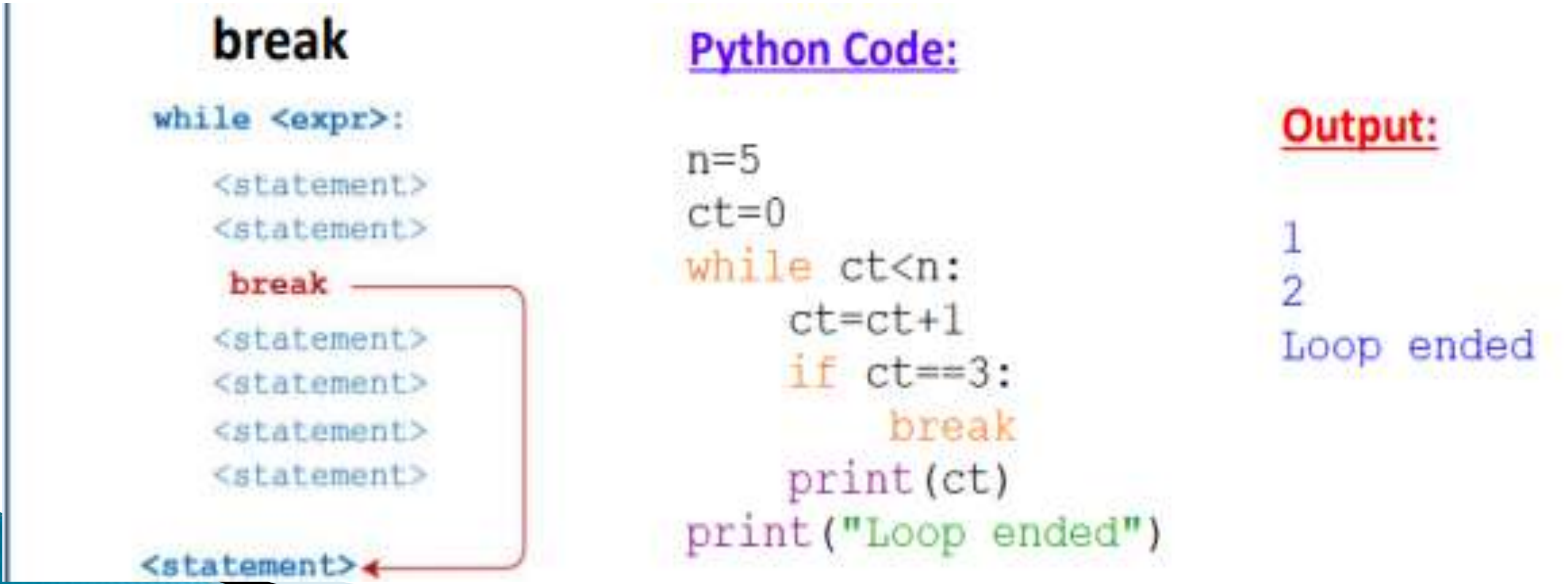
```
n=5
n=4
```

Break statements

Break

▪It immediately terminates a loop entirely.

▪Program execution proceeds to the first statement following the loop body.



break

```
while <expr>:
    <statement>
    <statement>
    break
    <statement>
    <statement>
    <statement>
    <statement>
<statement>
```

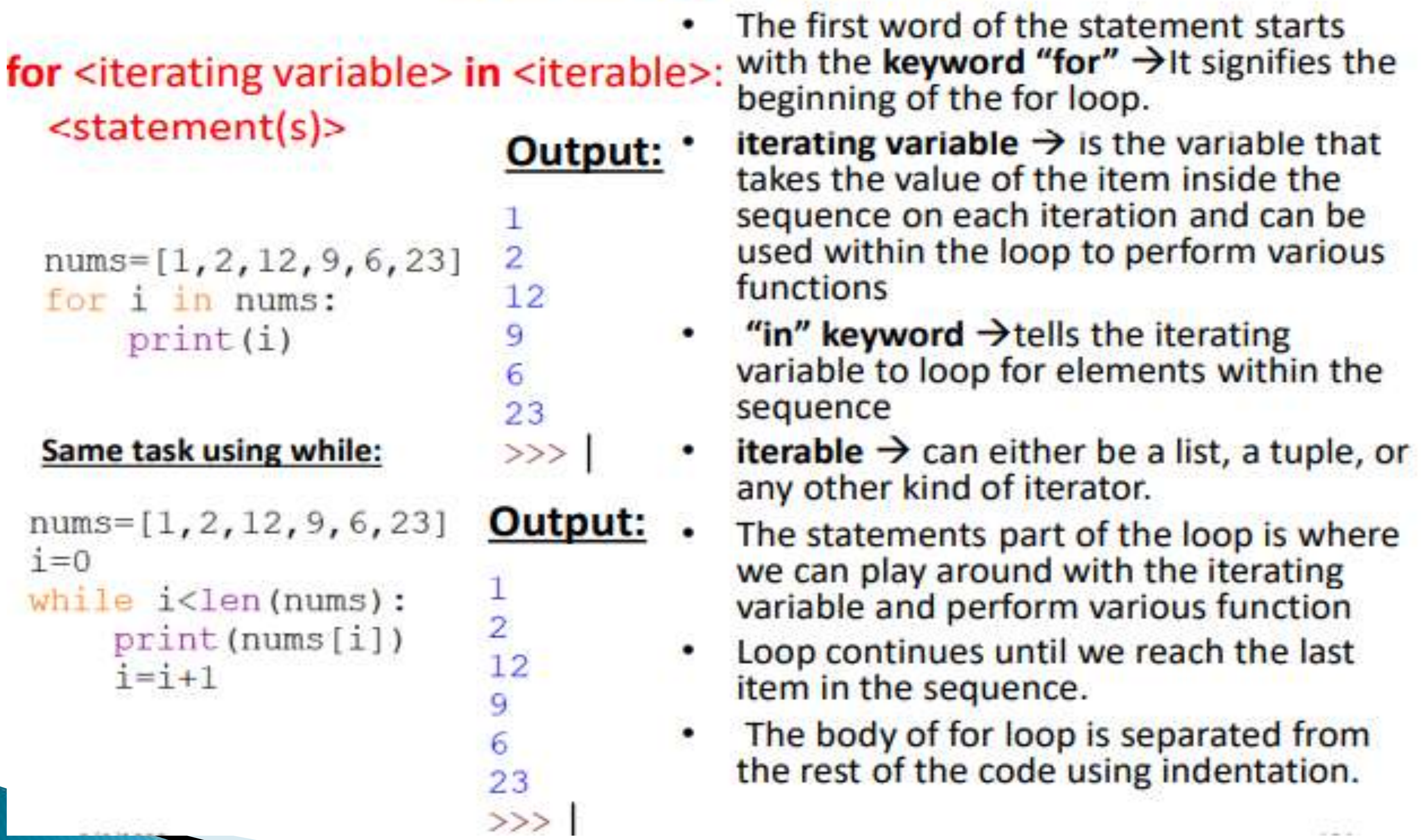
Python Code:

```
n=5
ct=0
while ct<n:
    ct=ct+1
    if ct==3:
        break
    print(ct)
print("Loop ended")
```

Output:

```
1
2
Loop ended
```

for loop Cont’d



for <iterating variable> **in** <iterable>:
    <statement(s)>

```
nums=[1,2,12,9,6,23]
for i in nums:
    print(i)
```

Output:

```
1
2
12
9
6
23
>>> |
```

Same task using while:

```
nums=[1,2,12,9,6,23]
i=0
while i<len(nums):
    print(nums[i])
    i=i+1
```

Output:

```
1
2
12
9
6
23
>>> |
```

- The first word of the statement starts with the **keyword “for”** →It signifies the beginning of the for loop.
- **iterating variable** → is the variable that takes the value of the item inside the sequence on each iteration and can be used within the loop to perform various functions
- **“in” keyword** →tells the iterating variable to loop for elements within the sequence
- **iterable** → can either be a list, a tuple, or any other kind of iterator.
- The statements part of the loop is where we can play around with the iterating variable and perform various function
- Loop continues until we reach the last item in the sequence.
- The body of for loop is separated from the rest of the code using indentation.

for loop Cont'd

Using the for loop to iterate over a Python list:

```
>>> for k in [1,2,3,4]:
        print(k)


1
2
3
4
```

```
>>> for k in ['Apple','Guava','Orange']:
        print(k)


Apple
Guava
Orange
```

Using the for loop to iterate over a Python tuple:

```
>>> for k in (1,2,3,4):
        print(k)


1
2
3
4
```

Using the for loop to iterate over a Python String:

(Print individual letters of a string using the for loop)

```
>>> for ch in 'Apple':
        print(ch)


A
p
p
l
e
```

The Built-in range Function

- Python provides a built-in **range function** that can be used for generating a sequence of integers that a for loop can iterate over
- range(10) will generate numbers from 0 to 9 (10 numbers).
- **Syntax:**

  range(start, stop,step_size)

  - ➢ The values in the generated sequence include the starting value, up to *but not including* the ending value
  - ➢ If start not mentioned default is 0
  - ➢ step_size defaults to 1 if not provided.
- range(i,j)→i,i+1,.....j-1
- range(j)→0,1,.....j-1 [automatically starts from 0]
- range(i,j,k)→i,i+k,.....i+nk [increments by k]

Python for loop with range() function

```
>>> for k in range(0,11):
        print(k)


0
1
2
3
4
5
6
7
8
9
10
```

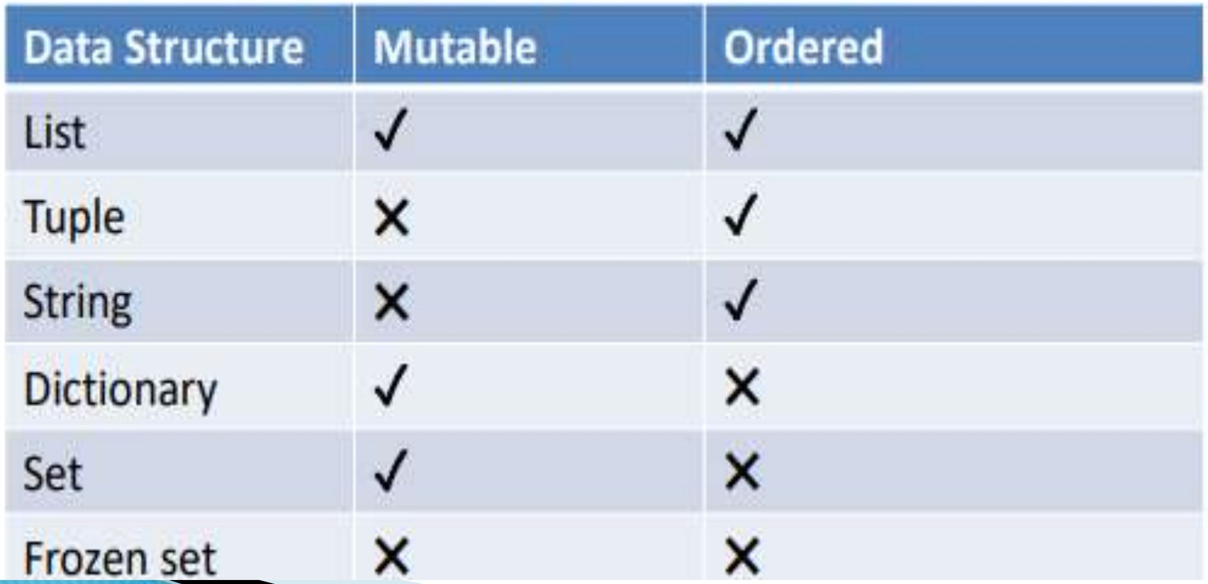
```
>>> for i in range(5):
        print(i)


0
1
2
3
4
```

```
>>> for x in range(2,10,2):
        print(x)


2
4
6
8
```

```
>>> for y in range(10,21,22):
        print(y)


10
```

DATA STRUCTURES

A computer is a programmable data processor that accepts input and instructions to process the input (program) and generate the required output



Data Structure	Mutable	Ordered
List	✓	✓
Tuple	✗	✓
String	✗	✓
Dictionary	✓	✗
Set	✓	✗
Frozen set	✗	✗

What is List in Python?

- A list is a **linear data structure** meaning that its elements have a linear ordering. i.e. there is a first element, a second element, and so on (similar to an array in other programming languages but more versatile)
- The values in a list are called **items** or sometimes **elements**.
- Lists are denoted by comma- separated list of elements within square brackets

  Eg. lst=[1,2,3]
- A list can even have another list as an item ie. Nested list

  Eg. Lst=["TV",[8,4,8],['a']]
- Empty list is denoted by an empty pair of square bracket []
- List in python use zero based indexing
- Elements are accessed by using an index value within square bracket
- Lst=[1,2,3] lst[0]→1 Access first element

  lst[1]→2 Access Second element

  lst[0]→1 Access Third element

CREATING LIST

1)The simplest is to enclose the values in square brackets [] :

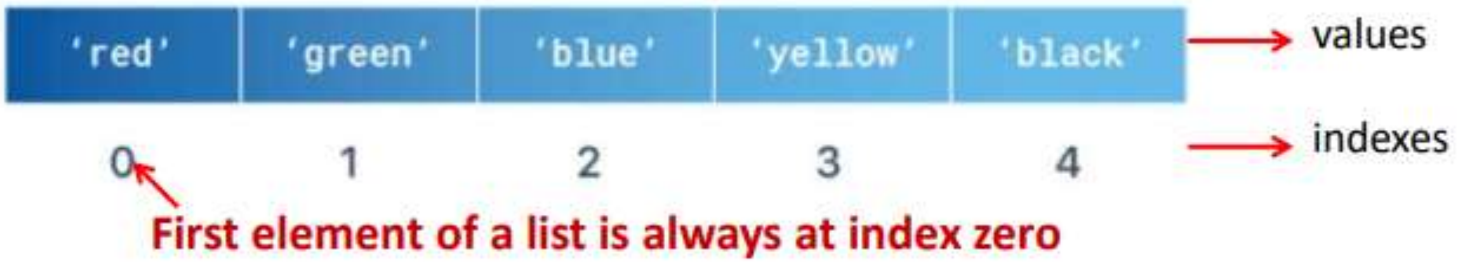
- L = [1, 2, 3] # A list of integers
- L = ['red', 'green', 'blue'] # A list of strings
- L = [1, 'abc', 1.23, (3+4j), True] # A list of mixed datatypes
- nlst=[1,2,['a','b'],[6,7.8,9.2]] #Nested List ・

2) A list containing zero items is called an empty list and we can create one with empty brackets []

- L = [] # An empty list

Finding an Item- Access List Items by Index

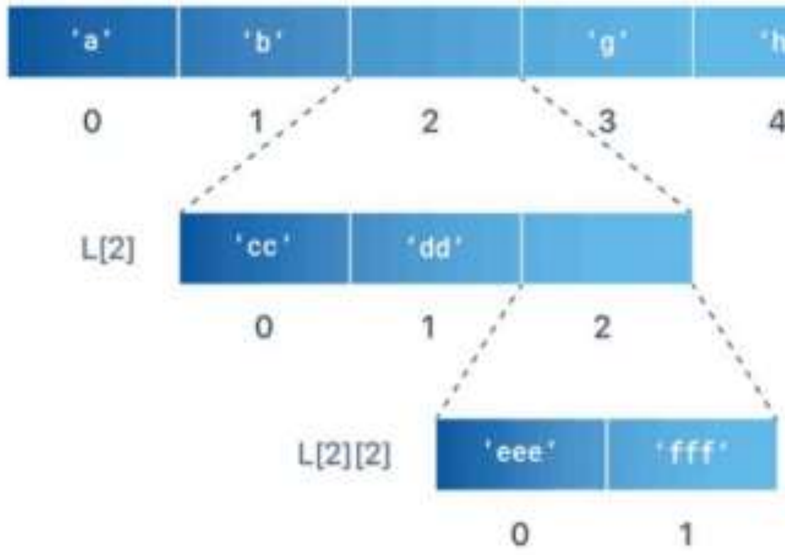
lst = ['red', 'green', 'blue', 'yellow', 'black']



```
>>> lst=['red', 'green', 'blue', 'yellow', 'black']
>>> lst
['red', 'green', 'blue', 'yellow', 'black']
>>> lst[0]
'red'
>>> lst[3]
'yellow'
```

Finding an Item/Access items in nested List

nlst= ['a', 'b', ['cc', 'dd', ['eee', 'fff']], 'g', 'h']



We can access individual items in a nested list using multiple indexes

The first index determines which list to use, and the second indicates the value within that list

```
>>> nlst= ['a', 'b', ['cc', 'dd', ['eee', 'fff']], 'g', 'h']
>>> nlst[0]
'a'
>>> nlst[2]
['cc', 'dd', ['eee', 'fff']]
```

```
>>> nlst[4]
'h'
>>> nlst[2][0]
'cc'
```

```
>>> nlst[2][2]
['eee', 'fff']
>>> nlst[2][2][1]
'fff'
```

Remove in list

To Remove an Item by Index:

1)pop() method: ・If we know the index of the item to be deleted we can use pop() method

・It modifies the list and returns the removed item.

・If no index is specified, pop() removes and returns the last item in the list.

2) del keyword :

・If we don’t need the removed value, we can use this.

Remove an Item by Value:

3)remove() method :

・If we are not sure where the item is in the list use remove() method

・If more than one instance of the given item is present in the list, then this method removes only the first instance.

Find List Length

- To find the number of items in a list, use **len()** method.

```
>>> lst=[1,5,6,7]
>>> len(lst)
4
```

Check if item exists in a list

- To determine whether a value is or isn’t in a list, we can use **in** and **not in** operators with if statement.

```
>>> lst=[1,2,3,4]
>>> 2 in lst
True
>>> 6 in lst
False
>>> 5 not in lst
True
>>> 4 not in lst
False
```

Looping Through a List

There are different ways to iterate over a list in Python:

- Using Python range() method
- By using a for Loop
- By using a while Loop
- List Comprehension

Looping Through a List Cont'd

- **Using Python range() method**
  - range() method can be used in combination with a for loop to traverse and iterate over a list in Python.

```
lst = [10, 50, 75, 83, 98, 84, 32]
for x in range(len(lst)):
   print(lst[x], end=" ")
```

Output:

```
10 50 75 83 98 84 32
>>>
```

- **By using a for Loop**
  - Python for loop can be used to iterate through the list directly

```
lst = [10, 50, 75, 83, 98, 84, 32]
for x in lst:
  print(x, end="  ")
```

Output:

```
10  50  75  83  98  84  32
>>>
```

Looping Through a List Cont'd

- **By using a while Loop**
  - Python while loop can also be used to iterate the list in a similar fashion as that of for loops.

```
lst = [10, 50, 75, 83, 98, 84, 32]
x = 0
while x < len(lst):
   print(lst[x],end="  ")
   x = x+1
```

Output:

```
10  50  75  83  98  84  32
>>>
```

List Comprehension Cont’d

- We can also create more advanced list comprehension which include a conditional statement on the iterable

```
newList = [expression for var in iterable if_clause]
```

This is equivalent to:

```
newList = []
for var in iterable:
   if conditional:
         newList .append(expression(var))
```

```
even=[x for x in range(11) if x%2==0]
print(even)
```

Output:

```
[0, 2, 4, 6, 8, 10]
>>>
```

List Comprehension Cont'd

```
addList=[x+y for x in [10,30,50] for y in [20,30,60]]
print(addList)
```

Output:

```
[30, 40, 70, 50, 60, 90, 70, 80, 110]
```

```
addList=[x+y for x in [10,30,50] for y in [20,30,60] if x!=y]
print(addList)
```

Output:

```
[30, 40, 70, 50, 90, 70, 80, 110]
```

```
x="asd235671jh"
sq=[2**int(n) for n in x if n.isdigit() if int(n)<=4]
print(sq)
```

Output:

```
[4, 8, 2]
```

THANK YOU ☺