* Introduction to python :-

1. python is popular language for programming.
2. It was developed by Guido van Rossum in 1991.
3. It is used for web development, software development Mathematics and system scripting.

* Advantages for python :-

1. python works on different platforms (windows, Mac, Linux, etc)
2. Simple syntax
3. Concise code compared to other languages.
4. Immediate execution with interpreter system.
5. It supports procedural, object-oriented and functional programming paradigms.

* what python can do :-

1. web application development
2. workflow automation.
3. Database comectivity and file manipulation
4. Handling big data and complex mathematics.

* python Indentation :-

1. The indentation of python is very important.
2. python use indentation to indicate block of code.
3. indentation refers to the space at the beginning of code line.

* python comments

1. comments can be used for explain the code
2. comments start with a #, python will that line.

Examples:-

1. single line comment:-

```
# this is comment
```

2. Multiline comment :-

* there is no multiline comment

Eg:-
```
# this is comment
# this is comment
# this is comment
```

3. Multistring comment :-

* It is start and ends with Triple quotes.

Eg:-
```
'''
This is comment
More than just one line
'''.
```

* Variables

1. Variables are containers for storing data values.
2. Variable can be short names (like x, like y) or more descriptive name (Name, age).

Rules for python variables:-

→ A variable name must start with a letter or underscore.

→ A variable name cannot start with a number.

→ A variable can contains alpha-numeric and underscore (A-z, 0-9, -)

→ Variable names are case-sensitive (age, Age, AGE are three different variables)

→ python variable cannot of any of python keywords.

Examples:-

```
x = 5
x, y, z = 'orange', 'Banana', 'apple'
x = y = z = "Apple"
```

* unpack collection

```
Fruits = ['Apple', 'orange', 'Banana']
x, y, z = Fruits
```

Global variable:-

* A variable can created outside of the function know as G.V
* Global variable can use both and everywhere inside and outside of function. [Note:- use global keyword while create inside the function]

* python datatypes

Datatype	Example	
str	x = "Anil"	→ Text type
int	x = 20	Numeric types
Float	x = 20.09	
complex	x = 1j	
list	x = ['Apple', 'Banana']	Sequence types
tuple	x = ('Apple', 'Banana')	
Range	x = range(5)	
dict	x = {'Name': 'Anil'}	→ Mapping type
set	x = {'Anil'}	Set types
Frozenset	x = Frozenset({'Ram'})	
bool	x = True	→ Boolean type
bytes	x = b"Hello"	Binary types
bytearry	x = bytearry(5)	
Memoryview	x = Memoryview(bytes(5))	
NoneType	x = None	→ None type

* python Casting:

Integers	Floats	Strings
x = int(1) → x = 1	x = Float(2) → x = 2.0	x = str(2) → x = '2'
x = int(2.3) → x = 2	x = Float(2.5) → x = 2.5	x = str(2.5) → x = '2.5'
x = int("20") → x = 20	x = Float("15") → x = 15.0	x = str("K") → x = 'K'

* Python Strings

* Assign string to variable

A = " Hellow"

* Assign multiline string to variable

x = """
sample Text of string
"""

* String length

*↔ To get the length of string, use the "len()" function

Eg:- Name = "Anil"
Result = len(Name) → Result = 4

* Check string

*↔ To check if a certain character or phrase is present in a string, we can use keyword "in"

Eg:- Name = "Anilkumar"
Result = "Anil" in Name → Result = True

*↔ To check if a certain character or phrase is not present in a string, we can use keyword 'Not in'

Eg:- Name = "Anil kumar"
Result = "Anil" not in Name → Result = False

* Slicing strings (syntax :- variablename[start:stop:step]

* you can return range of characters using slice index
* specify start index and end index separated by colon

Eg:- Name = "Anil kumar"

output = Name[1:4] → output = nil

1. Slice from start

→ By leaving out the start index, give end index alone

Eg:- Name = "Anil kumar"

output = Name[:4] → output = "Anil"

2. Slice From the End

→ By leaving out the end index, the range will go till end give start index alone.

Eg:- Name = "Anil kumar"

output = Name[5:] → output = kumar

3. Slice with step

→ Allows you to skip elements in the step sequence

Eg:- list = [1,2,3,4,5,6]

output = list[::2] → output = [1,3,5]

4. Reverse the string using slice

Eg:- Name = "Anil"

output = Name[::-1] → output = "linA"

* Modify Strings

1. Upper Case :-

→ The upper() method returns the string in uppercase

Eg:- Name = "Anil"

output = Name.upper() → output = ANIL

2. Lower Case :-

→ The lower() method returns the string in lowercase

Eg:- Name = "Anil"

output = Name.lower() → output = anil

3. Capitalize :-

→ The capitalize() method returns the first character of string is upper case.

Eg:- Name = "anil kumar"

output = Name.capitalize() → output = "Anil kumar"

4. Remove whitespace :-

→ whitespace is the before and/or after the actual text.

→ The strip() method removes any whitespace from the begining and end.

Eg:- Text = " Hellow world " (white space)

output = Text.strip() → output = "Hellow world"

5. Replace string

→ The replace() method replaces a string with another string

```
Name = "Anil kumar"
output = Name.replace("kumar", "Reddy")
output = "Anil Reddy"
```

6. Split string:-

→ The split() method returns a list. where the text between the specified separator becomes list items.

→ The split method splits the string into substrings. if it find instances of the separator.

Eg:-
```
Text = "Anil, kumar, Reddy"
output = Text.split(",")
output = ['Anil', 'kumar', 'Reddy']
```

7 Concatenate strings:

→ To concatenate or combine two strings you can use + operator

Eg:- Merge variable x with variable y into variable c

```
x = Anil
y = Kumar
z = x+y  ⟶  z = "Anil kumar"
```

8. Format string

→ Use the format() method to insert numbers into string.

Eg:- Name = "Anil"

Age = 23

output = "My name is {} and I am {} years old."

.format (Name, Age)

output = "My name is Anil and I am 23 years old.

* F-strings:-

→ F-string provide more readable to format strings.

Eg:- Name = "Anil"

Age = 23

output = f"I am {Age} years old and My {Name is {Name}

output = I am 23 years old and My Name is Anil.

* String Methods

Method	Description
Capitalize()	→ Converts the first character into uppercase.
casefold()	→ converts string to lower case.
center()	→ Returns a centered string.
count()	→ Returns the number of times occurs a specified value in a string.
encode()	→ Return encode version of string
endswith()	→ Return True if string ends with specified value
expandtabs()	→ set Tab size of the string.
find()	→ search specified string and Returns position
Format()	→ Formats specified values in a string.
format_map()	→ "
index()	→ search specified value and Retuns position
isalnum()	→ Returns True if All characters of string is Alphanumeric.
isalpha()	→ Returns True if all characters are Alphabet.
isascii()	→ Returns True, if all characters are ascii characters
is decimal()	→ Returns True, if all characters are decimal
is digit()	→ Returns True if all characters are digits.
isidentifier()	→ Returns True, if the string is an identifier.
islower()	→ Returns True, if all the characters are lowercase.
isnumeric()	→ Returns True, if all the characters are numbers.
is printable()	→ Returns True, if all characters are printable.

isspace() → Returns True, if all characters are space.

istitle() → Returns True, if string follow the rules of title.

isupper() → Returns True, if all characters are uppercase.

join() → Join the elements of an iterable to end of the string.

ljust() → Returns a left justified version of string

lower() → converts string into lower case

lstrip() → Returns left trim version of string

maketrans() → Retuns a translation table into used in translator

partition() → Returns a tuble where string parted into 3 parts.

replace() → Returns string, where specified value replaced to string.

rfind() → search string for specified value, Returns last position

rindex() → "

rjust() → Returns Right justified version of string

split() → splits the string at the specified separator.

splitlines() → splits the string at line breaks and retuns a list

startswith() → Retuns true. if the string starts with specified value.

strip() → Returns a trimmed version of string

swapcase() → swaps cases, lower case becomes uppercase and vicevers

title() → converts the first character of each word into upperca

translate() → Returns a translated string

upper() → converts a string into upper case

zfill() → Fill the string with specified value of 0 values at begi

* Python Booleans

→ Booleans represent one of two values : True or False

→ Most values are True, if it has some sort of content

→ Any string is True, except empty string

→ Any number is True, except 0.

→ Any list, tuple, set and dict are True, except empty ones

Eg:- * The following will return False

```
bool(False)   bool(0)   bool(())   bool({})
bool(None)    bool("")  bool([])
```

* Python keywords :

1. as
2. assert
3. await
4. and
5. async
6. True
7. False
8. def
9. class
10. return
11. try
12. except
13. finally
14. if
15. elif
16. else
17. For
18. while
19. break
20. continue
21. pass
22. not
23. or
24. is
25. in
26. global
27. del
28. raise
29. with
30. lambda
31. None
32. import
33. From
34. nonlocal
35. yeild

* Python Operators

→ operators are used to perform operations on variables & values

→ python devides the operators in the following groups :-

1. Arithmetic operators
2. Assignment operators
3. Comparison operators
4. logical operators
5. Identity operators
6. Membership operators
7. Bitwise operators.

1. Arithmetic operators :-

→ Arithmetic operators are used with numeric values to perform common mathematical operation

operator	Example	Name
+	x+y	Addition
-	x-y	subtraction
*	x*y	Multiplication
/	x/y	Division
%	x%y	Modulus
**	x**y	Exponentiation
//	x//y	Floor devision

2. Assignment operators

→ Assignment operators are used to assign values to variables.

operator	Example	
=	x = 20	
+=	x += 20	→ x = x + 20
-=	x -= 20	
*=	x *= 20	
/=	x /= 20	
%=	x %= 20	
//=	x //= 20	
**=	x **= 20	
&=	x &= 20	
|=	x |= 20	
^=	x ^= 20	
>>=	x >>= 20	
<<=	x <<= 20	

3. Comparison operators

→ Comparison operators are used to compare two value

operator	Example	Name
==	x == y	equal
!=	x != y	Not equal
>	x > y	less than
<	x < y	greater than
>=	x >= y	less than or equal to
<=	x <= y	Greater than or equal to

4 Logical operators

→ Logical operators are used to combine conditional statements.

operator	Example	Description
and	x<5 and x<10	both conditions Need to be True
or	x<5 or x>1	Atleat one condition Need to be True.
not		Reverse the result

5. Identity operators

→ Identity operators are used to compare the objects not if they are equal, but if they are actually the same object and with same memory location

operator	Example	
is	x is y	Returns True, if both variables are same object
isnot	x is not y	Returns True, if both variables are not same ob

6. Membership operators

→ Membership operators are used to test if sequence present in object.

operator	Example	description
in	x in y	if present True
not in	x not in y	if not present True

7. Bitwise operators

→ Bitwise operators are used to compare (binary) numbers.

operator	Name	Example	Description
~	NOT	~x	invert all the bits
&	AND	x & y	set 1 if both bits 1
|	OR	x | y	set 1 if any one bit 1
^	XOR	x ^ y	set 1 if only one bit is 1

* Python Collections

→ There are four collection data types in the python.

1. List → It is ordered and changeable, allows duplicates
2. Tuple → It is ordered and unchangeable, allows duplicates
3. set → It is unordered, unchangeable and unindexed, Don't allow duplicates.
4. Dictionary → It is ordered and changeable, NO duplicates

1. List :-

Note :- The first item has index 0.

* List Methods :-

Method	Description
append()	Add element at the end of list
clear()	Remove all elements from list
copy()	Returns a copy of list
count()	Returns count of specified value
extend()	Add elements of list, to the end of current lis
index()	Return index of first element
insert()	Add element at the specified position
pop()	Removes element at specified position
remove()	Remove item with specified value.
reverse()	Reverse the items with of list
sort()	

* Access items

list = ["Apple", "orange", "banana"]

1. Find first element

First element = list[0]

2. Find last element

Last element = list[-1]

3. Get range of elements

Range of elements = list[1:3]

* change item value

→ To change the value of specific item, refer to index number

list = ["Apple", "banana", "orange"]

list[1] = "grapes"

list = ["Apple", "grapes", "orange"] → Result.

* Add List items

1. Append items

→ To add an item to the end of the list, used append() method.

List = ["Anil", "Hari"]

list.append("vamsi")

list = ["Anil", "Hari", "vamsi"] → Result

2. Insert items

→ To insert a list item at a specified index. use insert() method

```
List = ["Ram", "Kiran", "Hari"]
list.insert(1, "Anil")
list = ["Ram", "Anil", "Kiran", "Hari"] → Result
```

3. Extend list

→ To append elements from another list to the current list, use extend() method.

```
list1 = ["Sumanth", "Bharath"]
list2 = ["Anil", "Hari"]
list1.extend(list2)
list1 = ["Sumanth", "Bharath", "Anil", "Hari"] → Result
```

* Remove list Items

1. Remove specified item

```
List = ["Anil", "Hari"]
list.Remove("Anil")
list = ["Hari"] ⟶ Result
```

2. Remove specified index

```
List = ["Anil", "Hari", "Vamsi"]
list.pop(1)
list = ["Anil", "Vamsi"] ⟶ Result
```

3. using del keyword

* Remove the specified index

```
List = [ "Anil", "Hari", "praveen"]
del list [2]
List = ["Anil", "Hari"]  ⟶ Result
```

* delete entire list

```
del list
```

4. Clear the list

→ the list still remains, but is has no content.

```
List = [ "Apple", "orange"]
list. clear ()
list = [ ]  ⟶ Result.
```

* Sort list Items

1. Sort the list (Ascending order)

```
List = [5, 10, 3, 6]
List. sort ()
list = [3, 5, 6, 10]  ⟶ Result
```

2. Sort the list (Descending order)

→ TO sort descending order using key argument reverse = True

```
list = ["Apple", "orange"]
list. sort (reverse = True)
list = ["orange", "Apple"]
```

3. Case sensitive sort

→ By default the sort() method is case sensitive, resulting in all capital letters being sorted before lower case letters.

```
List = ["hari", "Vamsi", "anil"]
List.sort()
list = ["Vamsi", "anil", "hari"]  ⟶ Result
```

4. Case insensitive sort

→ if you want a case-insensitive sort function, use str.lower as a key function

```
List = ["hari", "Vamsi", "anil"]
List.sort(str.lower)
list = ["anil", "hari", "Vamsi"]  ⟶ Result
```

* Reverse order

→ Reverse the current sorting order of the elements

```
List = ["Anil", "Ram", "Somu"]
List.reverse()
List = ["Somu", "Ram", "Anil"]  ⟶ Result
```

* Copy a list

→ you cannot copy simply by tying list1 = list2, whatever you change bothe list it will change.

1. list2 = list1.copy()
2. list2 = list(list1)

2. Tuple

* Customize sort function

→ we can also customize your own function by using the keyword argument key = function

Example:-

* sort the list based on how close the number to 50

```
def myfun (n)
    return abs(n-50)

list = [100, 50, 65, 82, 23]

list.sort (key = myfun)

list = [50, 65, 23, 82, 100]  → Result.
```

* list Comprehension

→ with list comprehension you can do all that with one li

```
Fruits = ["apple", "banana", "cherry"]

Newlist = [x for x in fruits if "a" in x]

New list = ["apple", "banana"]  → Result.
```

2. Tuple

→ Tuples are used to store multiple items in single variable

→ Tuples are written in round brackets.

* Access the Tuple items

```
Tuple = ("Anil", "Hari", "Ram")
first item = Tuple[0]
First item = "Anil"  ⟶ Result
```

* change Tuple values

→ once a tuple is created, you cannot change its values

Tuples are unchangeble or immutable.

→ we can convert the Tuple into list, change the list and convert list back to tuple.

Example:-

```
Tuple = ("Anil", "Hari")
list = list (Tuple)
list[1] = "Vamsi"
New_Tuple = tuple(list)
New_Tuple = ("Anil", "vamsi")  ⟶ Result
```

Note:- Same procedure for both add and remove also

3. Sets

→ Sets are written with curly brackets

→ set is a collection which is unordered, unchangeable and unindexed. duplicates not allowed.

* Access Items

→ you cannot access items in a set, by refering to an index or a key

→ But you can loop through the set items using for loop or ask if specified value is present in a set By using 'in' keyword.

Example:-

```
set = {"Anil", "Hari"}
```

1.
```
For x in set
    print (x)
```

2.
```
if "Anil" in set
    print ("Anil")
```

* Change set:-

Note:- once set created we cannot change items, but we can add items.

* Add Items

→ TO add onse item to set used Add() method.

```
set = {"Apple", "orange"}
set.add ("banana")
set = {"Apple", "orange", "banana"}
```
→ Result

* Add sets or Any Iterable

→ To add items from another set or anoter iterable to current set, use update() method.

Example:-

① set1 = {"Ram", "Logesh"}

set2 = {"Arun", "Suman"}

set1.update(set2)

set1 = {"Ram", "Logesh", "Arun", "Suman"} → Result

② set1 = {"Anil", "Hari"}

list1 = ["Vamsi", 'Bharat']

set1.update(list1)

set1 = {"Anil", "Hari", "Vamsi", "Bharat"}

* Remove Items from set

→ To remove an item in a set, use the remove() or discard() method.

Note:- 1. if the item to remove does not exit, remove() will raise error.

2. if the item to remove doesnot exit, discard() will not rais error