PYTHON. PROGRAMMING.

→ Adv / features of python :-

* High level p. lang.
* Interpreted p. lang. → execute line by line code.
* Highly portable → wrk any platform / software
* Extensible → combine module of other lang.
* Dynamically typing → assume the type of datatype.
* GUI (p) and databases support → coded to many libraries
* Cross platform
* Support other lang.
* Easy to code and Read.
* open source and free → which means that any one can create an contribute to its development.
* object oriented approach → class

→ IDLE (Integrated Developmnt and Learning Environment).

Has 2 types —

(1) Shell window
(2) ~~cursor~~ Editor window

→ Menues →

I (1) File menu → (1) New file
(2) open
(3) Resend files
(4) open module
(5) class browser → show ()s, classes and methods in the crnt editor file / shell
(6) path browser → show sys.path directories, modules, ()s, clses and methods in a pre str.

(7) Save
(8) Save as
(9) Save copy as
(10) Print
(11) Close
(12) Exit.

II Edit Menu :-
(1) Undo → Undo the last change to Crnt window
(2) Redo → Redo the last undone change to the crnt window.
(3) Cut
(4) Copy
(5) Paste
(6) Select all
(7) Find (8) Find in files (9) Replace
~~(10) For (11) Format menu~~

III Format Menu :-
(1) Indent Region → Indent 4 space → Tab
Shift selected lines right by the indent width.
(2) Dedent Region → Shift selected lines left by the indent width. (default 4 space)
(3) Comment out Region → Insert ~~# #~~ # # in front of selected lines.
(4) Uncomment Region → Remove leading # or # # from selected lines.
(5) New indent width → open a dialoge to change indent width.

IV Run Menu :- (Editor window only)
(1) Run Module → Do check module, if no error, Restart the Shell to clean the environment, then execute the module, output is displayed in the Shell window.
(2) Check module → check the syntx of module crntly open in the editor window.
(3) Python Shell → open / wake up the Python Shell window.

V Shell menu → [Shell window].
(1) View last restart :-
Scroll the Shell window to the last Shell restart.
(2) Restart shell :- restart the shell to clean the environment.
(3) previous history.
(4) New history :- cycle through later cmnts in history which match the crnt entry.
(5) Intrupt execution :- Stp the running prgm.

* Debugg :- ~~Err~~ when activated code entered in shell / run from an editor will run under the debugger. [Shell window].

* Python Cmnts :-
eg → print ("Hello world")
This is a cmnt. → Inline cmnt.

Cmnts
Inline / multyline

used to prevent execution in testing code. In Python # indicates a cmnt.

* Cmnts can also be placed at the end of a line → Inline cmnts.

eg:- print ("Hello") } 1 cmnt.
This is a cmnt.

This is a cmnt. } cmnt.
print ("Hello")

* multyline cmnts:-

① To add a multyline cmnt you have to insert a # for each line
eg → # comment
more just 1 line
2 comment.
print ("hello").

② you can use a multyline string,
eg → """ more just 1 line
2 comment"""
print ("hello")

→ Python identifier :-
It is a name given to prgrm entities like cls, fns, variables, etc.
* Rules →
- Identifiers can be a combination of lowercase / uppercase letters, digits / _ .
- An identifier cannot begin with a digit
- Identifiers cannot have sple characters like !, @, #, %, $ ...
- Python keywords cannot be used as identifiers
- Identifier can be of any length.

→ Python keywords :-
Sple identifiers which are reserved and have a specific meaning.
eg → for, while, def, print, if, break, continue... int, float, True, False.

→ Variables :-
It is a name (identifier) that is associated with a value.
As the name implies, a variable is something which can change and therefore can assign different values during a prgrm execution.
A = 'Abhiram' → Str type
* x = 4
x = 'Jay' 4 → x = Jay & 'Jay'.
print(x) → Jay

Rules →
* A variables name must start with a letter, or Character.
* A V. name cannot start with a no.
* A V. name can only contain alpha numeric characters & _ . (A-Z, a-z, 0-9 & _)

* V. names are case sensitive (age, AGE, Age are 3 different variables).

eg → x = "awesome"
print("python is " + x)

python print statement is often used to output variables.
To combine both text and a variable, python uses the '+' character.

→ Datatypes :- which type of data are stored in a memory location.
It can be many types →

1) Numeric → int, float, complex.
2) String → str
3) Boolean → bool.
4) Sequence type → list, tuple, range.
5) set type → set
6) Binary type → bytes array, byte array.

→ Type () :-
You can get the datatype of any object by using type().

eg → x = 5
print(type(x)) → ~~Integer~~ <class int>
x = 1j
print(type(x)) → complex.

→ Boolean type () :- (T or F.)
Booleans represents 1 of 2 values → T or F.

eg → print(10 > 9) → T
print(10 == 9) → F

print(bool(15)) → T → byte → int so ✓
print(bool("hello")) → T → str → ✓.
print(bool()) → F.

* python evaluates almost any value to T if it has some sort of content.
* Any string is T, except empty strings.
* Any no. is T, except ~~empty~~ 0.
* Any list, tuple, set and dict are T, except empty once.

⇒ Binary type :-

(1) Bytes data type :- It represents a grp of byte nos. just like a array does. (Bytes array)
* A byte no. is any +ve integer from 0 to 255.
~~Bytes array can~~
eg → elements = [10, 20, 0, 40, 15]
x = bytes(elements)
print(x[0]). → 10
* we cannot modify / edit any elemnt in the bytes type array.

elements = [10, 20, 0, 40, 15]
x = bytes(elements)
for i in x:
print(i) → ~~10, 20, 0, 40, 15~~

→ 10
20
0
40
15

(2) Bytes array :-

It similar to bytes data type, the difference is that the bytes type array cannot be modified bt byte [] type can be modified

* eg → elements=[10,20,0,40,15]
x = bytes(element)
print(x[0]) → 10.

edit { x[0]=55 } elements replaced
x[1]=99

for i in x:
print(i) → 55
99
0
40
15

* E = [10,20,30].
x = byte(E)
x[0] = 100
print(x[0]) → 100

[0,1,2 index]

* list Data types :- []

* list in python are similar to []s in c / Java
* A list represents a grp of elemnts The main difference b/w a list and an array is that a list can store different types of elemnts bt an [a] can store only 1 type of elemnts.
* list can grow dynamically in memory bt the size of [a] is fixed & they can grow at run time

[size of [a] → fixed
" list → change]

Store multiple type of data { list → mutalele, tuple → immutalele.

* list are represented using [] & the elemnts are returned in [] separated by commas.
eg → list = [10,20,30,'Siva']
print (list)

* Tuple Data type :- ()

* A tuple is similar to list, it contains a grp of elemnts which can be different type.
* The elemnts in the tuple are separated by commas & enclosed in ().
where as list elemnts can be modified (mutalele) bt tuple cannot be modified (immutalele).
* eg → t1 = (10, 20, 30, 'Siva')
t1[0] = 99 → error (immutalele)

* Range Data type :-

* range Data type represents sequence of nos
* The nos in the range are not modifialele.
* used for repeating a for loop for a specific nos of times.
* eg → r = range(10)
for i in r:
print(i) → 0 1 2 3 4 5 6 7 8 9 (0→9)

set → modified
frozen set → no modification.

* Set Data types :- {} (no repeated values)

* Unordered collection of elemnts → Set.
not read repeated ones.
eg → s = {10, 20, 20, 30}
store 10, 20, 30.
* A set is an unordered collection of elemnts much like set in maths.
* The order of elemnts is not maintained in the sets.
* A set does not accept duplicate elemnts.
* 2 types → 1) Set
2) frozen set.

* ~~2 datatypes~~
* To create a set, one should enter the elemnts seperated by commas inside {}
eg → S = {10, 20, 30, 20}
print (S) → {10, 20, 30}
→ A = "Hello"
print (A) → {'H', 'e', 'l', 'o'}

(output can be comma seperated values)

* Update () → used to add elemnts into a set.
eg → S = {10, 20, 30, 30} [can be added at any position]
S.update([50, 60])
print (S) ⟷ {10, 20, 30, 50, 60}.

* frozen set () :- It is same as the set data type, the main difference is that elemnts in the set datatype can be modified where as, elemnts of frozen set cannot be modified.
* one can create a frozen set () by passing a set to frozen set (). as
eg → S = {50, 60, 70, 80, 90}
print (S)
fs = frozenset (S)
print (fs). → {50, 60, 70, 80, 90}

⇒ operators :- (o) (operations of operands.)

1) Arithmetic (o)
2) Assignment (o) +=, -=, *=, /=
3) Comparison (o) (relational & logic) ==, !=, <=, >=
4) Logical (o) AND, OR, NOT
5) Identity (o) is, is not
6) Membership (o) in, not in
7) Bitwise (o) (& OR, & AND, left shift.)

* Identity (o) :- used to combare the objects with the same memory location.
1) is
2) is not
1) is → Evaluates to true if the variables on either side of the operator point to the same object & false otherwise
2) is not → Evaluates to false if the

id() → memory location

Variables on the either side of the operator point to the same object & true otherwise.

eg → a = 10
b = 20
c = a → identity (a)
print (bool (id(a) == id(b))) → False
print (bool (id(a) == id(c))) → True

* Membership (o) :-
Python's membership (o) test for membership in a sequence such as strings, list & tuples.

1) in → Evaluates to true if it finds a variable in the specified seques & false otherwise

2) not in → Evaluates to if it does not finds a variable in the specified sequence & false otherwise.

eg → List = [1, 2, 3, 4, 5]
print (6 in List) → false.

~~* Control Statements :-~~

* Statements & expressions :-

• A statemnt is a unit of code that the python interpreter can execute.

• There are different types of statemnts in python, →
Assignment (s), return (s), break, for, while, if, import, continue, etc.

• It can be extended to multiple lines using (), [], ;, etc.

• \ - is used for continue the (s).

* An expression is a combination of values, variables, operators, and calls to (f)s that evaluates to a value.
eg → A.M
Boolean → true / false
logical.

* Input method :-

user inputs :
Python provides a built in () calling input function, that gets input from the key board.

Syntax →

```
x = input ("enter a name")
print ("hello ", +x)
```

output → : hallo enter a name.

* Type () :-

It returns the type of the specified object.
It is mostly used for debugging purpose.

Syntax → type. (exp).

eg →

```
count = 0
print ("type of count is", type
(count))
```

Control Statements.

* A control statement is a statement that determines the control flow of a set of instructions.

There are 3 fundamental forms of control that programming lang provides—

1) Sequencial control
2) Selection control → if-else, if, else
3) Iterative control → while loop, for loop.

1) Sequencial control :- It is an implicit form of control in which instructions are executed in the order that they are written.

2) Selection control :- It is provided by a control statmnt that selectively execute instructions.

3) Iterative control :- It is provided by an iterative control statmnt that repeatedly executes instructions.

* Control Str :- collectively a set of instructions & control statmnt controlling their execution → C. Str.

* Boolean expressions :- It is a exp that evaluate to one of 2 boolean

Values T/F.

eg → print (a>b) → T/F.

* Compound boolean exp :-
Single logical exp constructed by using comparison operators can be chained with the logic / boolean operators AND / OR / NOT.

eg → if (a>b and b>c)
if (a>b or b>c)

* Selection control statements / Decision making statmnts. → if, if-else, elif.

1) if statmnt →

Syntax → if expression :
statemnt(s).

eg →
num = int (input ("Enter a number :"))
if (num %2 == 0) :
print ("num", "is a even")

2) if-else →
Syntax → if exp :
statemnt
else :
statmnt

eg → n = int (input ("Enter no"))
if (n %2 == 0) :
print ("num", "even")
else :
print ("num", "odd").

3) Multiway selection :-
if - elif - else

Syntax →
if exp :
(s)
elif exp
(s)
.....
else :
(s)

eg → clr = input ("Enter colour"))
if (clr == 'V') or (clr == 'v') :
print ("you selected red")
elif (clr == 'I') or (clr == 'i') :
print ("you selected indigo")
.....
else :
print ("Invalid").

4) Nested if-else →

Sytax →
if condition c_1 :
(s_1)
if condition c_2 :
(s_2)
else :
(s_3)
else :
(s_4).

* Iterative Control Statements → for, while

It is control statement providing the repeated execution of a set of instructions.
2 types of iteration →

(1) Definite (I)	(2) Indefinite (I)
* Here the num of times the designated block will be executed is specified explicitly at the time the loop start.	Here the num of times the loop is executed is not specified explicitly in advance.
* eg → displaying num from 1 to 100.	eg → displaying num from 1 onwards ↓ And asking the user whether the user want to display the nxt.

I while statement :-

Syntx → while exp:
Statemnts.

eg → n=1
while (n<100):
print(n)
n=n+1.

→ 1 2 3 99

II for statemnt :-

Syntx → for i in range(n):
Statemnts. —①

Syntx → for variable in iterable: —②
Statemnt

in → for loop keyword
not membership (0).

eg → Fruits = ["Mango", "orange"]
for i in Fruits:
print(i)
end=" " → Mango orange. → Mango orange

① here i is called a loop counter
& n → num of times that the loop will execute the statemnt.

⇒ * range () :-

It is a built-in () in python.
The range(n) generates a sequence of n integers starting at 0.
It uses the value by 1 until it reaches n.
So the range(n) generates a sequence of num 0, 1, 2 ..... (n-1).

Syntaxe → for i in range(n):
print(i).

eg → for a in range(5):
print(a)

→ 0
1
2
3
4

[Default range index no is 0.]

* range (Start, Stop) → range syntx.

In this syntx, the range() uses the Start the value by 1 until it reaches the Stop value.

eg → for i in range(5,10):
print(i)

→ 5
6
7
8
9

* range (Start, Stop, Step) → range syntx.

eg → for ak in range(5,10,2):
print(ak)

→ 5
7
9

In this form you can specify the value that the range() should use.

=> Nested loop :- for, while.

(1) 2-level nested for loop → for loop

eg → for var in seq:
for jar in ak:
Statemnt(s)
Statemnt(s)

(2) 2-level nested while loop → while loop

eg → while exp:
while exp:
Statmnt(s)
Statemnt(s).

=> Infinite loops :-
It executes the block of Statemnts repeatedly until the user forces the prgm to quit.

eg → web server may need to continously check for incoming connections.

```
while True:
    print("Ak_programz")
```

To Stop this, use break or return Statemnts. or Sys.exit [import sys ...] embedded somewhere within its body. or Ctrl + c → to exit frm loop

[10, 20, 'Sh'] — list
(10, 10.20) — tuple.

03 : Strings, lists & tuples in python

* String operator → = [assecing]
* Slicing →
Accessing substring.
~~print~~ ("Hello, world")

b = "Hello, world"
print (b[2:5]) → llo,

String is immutable

→ Strings in python :-

a = "Hello_world"
print (a[1]). → e

* You can return a range of characters by using the slice syntax,
* Specify the start index & end index, seperated by a colon, to return a part of string → slicing.

b = "ak_programz"
print (b[0:10]) → ak_programz.

* Try to access a characters out of index range will ~~use~~ cause an error.
* Index must be a int expressions, use float / other type will results in type Error.
* python allows -ve indexing for its sequences.

* Index of -1 refers to the last item, -2 to 2nd last item & so on.
ak = "Hello, world!"
print (ak[-5:-2]) → ~~Traceback~~

→orld

w	o	r	l	d	!
-6	-5	-4	-3	-2	-1

ak [-5 : -2]

orl d

b = "ansarbb"
print (b[2:6:2])

s a r b

→ s a r b

~~[:]~~

* a = "hello"
print ([:]) → hello.

* del → ~~Strings~~ are deleted.

→ Delete / update a str :-

* Strings are immutable → means that elements of a str cannot be changed, once it has been assigned.
ak = 'hello'
~~print (ak[1])~~
ak[1] = 'h'.
print (ak) → type error.

* **Delete** :- 'del' keyword [entire str deleted]
Single char cannot be deleted.

You cannot edit / remove char from a str, editing the str entirely is possible using the keyword ~~del~~ del.

eg= a = 'hello'
print (txt)
del txt ⟶ name error.
print (txt)

* **Escape characters** :-
It is a back slash (\)
followed by the character you want to ins.
Escape sequences allow for including sple characters into str.
eg → newline ⟶ \n
backslash ⟶ \\
\' ⟶ Single Slash
\b ⟶ backspace
\t ⟶ horizontal tab
\v ⟶ vertical tab

* **String operators** :-

+	, * ,	[]	, [:] , in , not in
concat ↓ add values of either side of ().	repetition ↓ creates new str, concatinating multiple copies of the same str.	slice	rangeslice ↓ gives character from given indx.

[eg] + → a = 'hello'
b = 'python'.
c = a+b
print (c)

* → a = 'hello'
~~b = 'python'~~
a * 2
→ hellohello.

[] → a[1]
→ e

[:] → ~~a = [hk]~~

	0	1	2	3	4
a =	A	n	s	a	r

a [1:3] → ns

→ gives the characters from given range.

in → returns true if a character exist in the given str.
a = 'Hello'
print ('H in a') → True.

not in → returns true if a character does not exist in given str.
print ('d not in a') → True.

* **Build-in str methods** →

1) **capitalize()** → eg → a = 'ansar'
print (a.capitalize()) → Ansar
returns a str where the 1st character is upper case
Syntax → string.capitalize()

2) **center()** → It will center align the str using a specified character (space is deferact)
Syntax → str.center(length, character)

Ps → T/F.

3) Casefold() → returns a str where all the characters are lowercase.
* This method is similar to lower() method.
* But the casefold is stronger, more aggressive, meaning that it will convert more characters to lowercase & will find more matches when comparing 2 strings & both are converted using casefold method.

4) isalnum() → Returns True if all characters are alphanumeric → (A-Z, a-z, 0-9) spl characters X.
Syntx → str.alnum()

5) isalpha() → returns True if all characters are alphabets (A-Z, a-z)
Syntx → str.isalpha()

6) isdecimal() → 0-9
Syntx → str.isdecimal()

7) isdigit() → returns True if all characters are digits otherwise False.
Syntx → str.isdigit()

8) isidentifier → returns true if str is a valid identifier, otherwise F.
* A str is considered a valid identifier if it only contains alphanumeric letters (A-z & 0-9) or _).
* A valid identifier cannot start with a num or any spaces
Syntx → str.isidentifier()

9) islower() :-
This method returns True if all characters are in lowercase.
Syntx → str.islower()

10) isnumeric() :-
This method returns true if all characters in numeric. (0-9) otherwise false
Syntx → str.isnumeric()

11) isspace() :-
The is space method returns true if all the characters in string are whitespaces

Eg:- str.isspace()
str.isspace()

=> list operations :-

1) append() :-
Syntx → list.append(obj).
appends a past object into the existing list.
obj → object to be appended in the list.

[] → list

eg → L = ['AK programz', 'AK coding']
L.append('AK buildings')
print(L)

2) insert() :- append() only works for addition of elements at the end of list.
for addition of elemnt at desired position, insert() is used.
Unlike append() which takes only 1 argument, insert() requires 2 arg. (pos, value) pos → position.

Syntx → List.insert(pos, value)
pos → new specifying in which position to insert the value.
value → Required arg & can be element of any type. (str, no..)

eg → L = [1,2,3,4]
print("initial list", L)
L.insert(3,12)
print(L)

→ 1 2 3 4
1 2 3 12 4

3) count() :- Returns count of how many times obj occurs in list
Syntx → L.count(obj)

eg = L = [123, 'AK', 9, 'AK', 'C', 'AK']
print(L.count('AK')) → 3

4) list extend() :- It appends the contents of sequence to [].
L.extend(seq)
seq → [] of elements
L1 = ['phy', 'che', 'bio']
L2 = list(range(5))
L1.extend(L2) → phy, che, bio, 0, 1, 2, 3, 4

→ Tuple() :-

* Basic Tuple operations :-

1) Length :-
eg → th = ("apple", "banana", "cherry")
print(len(th)) → 3

2) Concatenation :-
You can use '+' operator to combine 2 tuples → ().
eg → print((1,2,3)+(4,5,6)) → 1,2,3,4,5,6

3) Repetition :-
Repeat operator (*) for repeating the elements of a given no. of times using the * operator.

eg → print("AK",*3) → (AKAKAK)

4) Deleting a tuple :-
The elements of a tuple cannot be change once it has been assigned that also means you cannot remove/delete an items from a tuple.
But deleting a tuple entirely is possible using the using the keyword "del".

eg = Ak = ("programz", "bk")
~~print(del(Ak))~~
del(Ak).

5) count() :- returns num of times a specified value occurs in a tuple.
Syntx → tuple.count(value)
eg → t = (1, 3, 7, 7)
print(t.count(7)) → 2

6) index() :- finds the 1st occurance of the specified values.
It rises an exception if the value is not found.
Sytx → tuple.index(value)
eg → t = (1, 3, 7, 8, 7, 5)
print(t.index(8)) → 3

[1, 2, 'str'] — list
(1, 10, 20) — tuple

03 : Strings, lists & tuples in python

* String operator → = [assecing]
* Slicing →

Accessing substring

~~print ("Hello, world")~~

```
b = "Hello, world"
print(b[2:5])   → llo,
```

String is immutable

→ Strings in python :-

```
a = "Hello_world"
print(a[1]).    → e
```

* You can return a range of characters by using the slice syntax,
* Specify the start index & end index, seperated by a colon, to return a part of string → slicing.

```
b = "ak_programz"
print(b[0:10])  → ak_programz.
```

* Try to access a characters out of index range will cause an error.
* Index must be a int expression, use float / other type will results in type Error.
* python allows -ve indexing for its sequences.

* Index of -1 refers to the last item, -2 to 2nd last item & so on.

```
ak = "Hello, world!"
print(ak[-5:-2])   → orld
```

w	o	r	l	d	!
-6	-5	-4	-3	-2	-1

ak[-5:-2] → orld

```
b = "ansary bb"
print(b[2:6:2])
```

s a r b

* [:]
* a = "hello"
  print([:]) → hello.
* del → all strings are deleted.

→ Delete / update a str :-

* Strings are immutable → means that elements of a str cannot be changed, once it has been assigned.

```
ak = 'hello'
~~print(ak[1])~~
ak[1] = 'h'.
print(ak)   → type error.
```

* **Delete** :- del keyword [entire str deleted]
Single char cannot be deleted.
You cannot delt / remove char from a str, dlting the str entirely is possible using the keyword ~~dlt~~ del.
eg= a = 'hello'
print (txt)
del txt ⟶ name error.
print (txt)

* **Escape characters** :-
It is a back slash (\)
followed by the character you want to ins.
Escape sequences allow for including sple characters into str.
eg → newline ⟶ \n
backslash ⟶ \\
\' ⟶ single slash
\b ⟶ backspace
\t ⟶ horizontal tab
\v ⟶ vertical tab

* **String operators** :-

+	, * ,	[]	, [:] , in , not in
concat → add values of either side of ().	repetition → creates new str, concatinating multiple copies of the same str.	slice	range slice → gives character from given indx.

[eg] + → a = 'hello'
b = 'python'.
c = a+b
print (c)

* → a = 'hello'
~~b = 'python'~~
a * 2
→ hellohello.

[] → a[1]
→ e

[:] → ~~a = a[1:]~~

	0	1	2	3	4
a =	A	N	S	A	R

a [1:3] → NS

→ gives the characters from given range.

in → returns true if a character exist in the given str.
a = 'Hello'
print ('H in a') → True.

not in → returns true if a character does not exist in given str.
print ('d not in a') → True.

* **Build-in str methods** →

1) capitalize () → eg → a = 'ansar'
print (a.capitalize()) → Ansar
returns a str where the 1st character is upper case
Syntax → string.capitalize()

2) center() → It will center align the str using a specified character (space is default)
Syntax → str.center(length, character)

Ps → T/F.

3) casefold() → returns a str where all the characters are lowercase.
* This method is similar to lower() method.
* But the casefold is stronger, more aggressive, meaning that it will convert more characters to lowercase & will find more matches when comparing 2 strings & both are converted using casefold method.

4) ~~is~~ isalnum() → Returns True if all characters are alpha numeric → (A-Z, a-z, 0-9) spl characters X.
Syntx → str.alnum()

5) isalpha() → returns True if all characters are alphabets (A-Z, a-z)
Syntx → str.isalpha()

6) isdecimal() → 0-9
Syntx → str.isdecimal()

7) isdigit() → returns True if all characters are digits otherwise false.
Syntx → str.isdigit()

8) isidentifier → returns true if str is a valid identifier, otherwise F.
* A str is considered a valid identifier if it only contains alphanumeric letters (A-Z & 0-9) or _).
* A valid identifier cannot start with a non or any spaces
Syntx → str.isidentifier()

9) islower() :-
This method returns True if all characters are in lowercase.
Syntx → str.islower()

10) isnumeric() :-
This method returns true if all characters in numeric. (0-9) otherwise false
Syntx → str.isnumeric()

11) isspace() :-
The is space method returns true if all the characters in string are whitespaces

Eg:- str.isspace()
str.isspace()

=> list operations :-

(1) append() :-
Syntx → list.append(obj).
appends a past object into the existing list.
obj → object to be appended in the list.

[] → list

eg → L = ['AK programz', 'AK coding']
L.append('AK buildings')
print(L)

2) insert() :- append() only works for addition of elements at the end of list.
for addition of elemnt at desired position, insert() is used.
Unlike append() which takes only 1 argument, insert() requires 2 arg. (pos, value) pos → position.

Syntx → List.insert(pos, value)
pos → new specifying in which position to insert the value.
value → Required arg & can be element of any type. (str, no.)

eg → L = [1,2,3,4]
print("~~Initial list~~", L)
L.insert(3,12)
print(L)

~~→ 1,2,3,4~~
→ 1 2 3 4
~~1,2,3,4,8,12~~
1 2 3 12 4

3) count() :- Returns count of how many times object occurs in list
Syntx → L.count(obj)

~~(1)~~

eg = L = [123, 'AK', 9, 'AK', 'C', 'AK']
print(L.count('AK')) → 3

4) list extend() :- It appends the contents of sequence to [].
L.extend(seq)
seq → [] of elements
L1 = ['phy', 'che', 'bio']
~~print(L.extend)~~
L2 = list(range(5))
L1.extend(L2) → phy, che, bio, 0, 1, 2, 3, 4

→ Tuple() :-

* Basic Tuple operations :-

1) Length :-
eg → th = ("apple", "banana", "cherry")
print(len(th)) → 3

2) Concatenation :-
You can give '+' operator to combine 2 tuples → (()).
eg → print((1,2,3)+(4,5,6)) → 1,2,3,4,5,6

3) Repetition :-
Repeat operator (*) for repeating the elements of a given num of times using the * operator.

eg → print("AK",*3) → (AKAKAK)

4) Deleting a tuple :-

The elements of a tuple cannot be change once it has been assigned that also means you cannot remove/delete an items frm a tuple.
But deleting a tuple entirely if possible using the using the keyword "del".

eg = Ak = ("programz", "bk")
del(Ak).

5) Count() :- returns num of times a specified value occurs in a tuple.
Syntx → tuple.count(value)
eg → t = (1, 3, 7, 7)
print(t.count(7)) → 2

6) index() :- finds the 1st occurance of the specified values.
It raises an exception if the value is not found.
Syntx → tuple.index(value)
eg → t = (1, 3, 7, 8, 7, 5)
print(t.index(8)) → 3

=> Python dict =

A dict is a collection of indexed changeable & unordered key:value pairs.

* In a dict, keys must be unique & stored in an unordered manner.
* eg → Ak = {'ab':15, 'cd':20}.
print(Ak)

* Accessing elements frm dict →

{'Ak': 'John'}
d = {'name': 'John', 'age':20, 'class':'BSc cs'}
print(d.get(name))
→ John.

* Indexing is used with other containing time to access values, dict uses keys.
* key can be used either inside [] or with the get method.
* Indexing →
a = {'name':'Ak', 'class':'BSc cs'}
print(d['name']) → Ak.

* The diff - while using get() it returns none instead of key error if the key is not found.

=> Dlt or remove elements frm dict :-

eg → Ak = {1:1, 2:4, 3:4, 4:2}
print(pop(3))

* You can remove a particular item in a dict by using pop()

① pop() key/position
② popitem() key value.
③ clear() on dict

This method removes an item with the provided key & returns the value.

* pop item() used to remove & return an arbitary item (key:value) from the dict.

* All the items can be removed at once using the clear().
You can also use the 'del' keyword to remove individual items of the entire dict itself.

eg →
S = {1:1, 2:4, 3:9, 4:16, 5:25}
print(S)

pop() → print(S.pop(4)) → {1:1, 2:4, 3:9, 5:25}
pop item() → print(S.pop item()) → {2:4, 3:9, 5:25}.

del → del S[2]. → {3:9, 5:25}.
remove all items → S.clear() → {}
del dict → del S. → error

→ Build-in () =

1) all() → returns true if all keys of dict are true (or if the dict is empty)

2) any() → returns T if any key of dict is T, if the dict is empty returns F.

3) len() → returns the length.

4) cmp() → Compares items of 2 dict.
(Compare)

5) sorted() → return a new sorted list of keys in dict.

eg → AK = {1:1, 2:4, 3:9}
print(len(AK)) → 3

→ Dict methods =

a) clear() = dict.clear().
b) copy() = copy of the dict.
dict.copy()
c) get() = returns value of specified key.
dict.get(key, value).

⇒ sets = {}.

* A set is an unordered & unindexed collection of items. (no indexing & slicing)
* Every element is unique (no duplicates) & must be immutable.
However the set itself is mutable you can add / remove items frm it.
* A set is created placing all elements inside {}.
* Sets can hve any no of elemts & they may be of any datatype.
eg → set = {1, 2, 3}.
print(set).
* set() → a str can be passed to set() as well.
eg → # set of chr frm str.
AK = set('AK programming')
print(AK).
* set(str) generates a set of chars in str.

→ Accessing & Changing Set =

* add() =
You can add a single element

* update() =
You can add multiple element.
It can take tuples, lists, str / other sets its arguments.
eg →
Set = {1, 3}
print (set)
add element → set.add(2)
print(set) → {1, 3, 2}
add multiple element → Set.update([2,3,4])
print(Set) → {1, 3, 2, 4}

→ Removing elements from set =

* A particular item can be removed from set using discard() & remove()
* The only diff - b/w 2 is that while using discard(), if the the item does not exist in the set, it remains unchanged, but remove() will raise an error in such condition.
* eg →
Set = {1, 3, 4, 5, 6}
print(Set.discard(4))
print(Set.remove(6))
→ {1, 3, 5}

union → |
Intersection → &
Difference → -

→ Set operations =

1) Set union → union of A & B is a set of all elements from both sets.
→ In python, union is performed using |.
→ Same can be accomplished using union().
→ eg ⇒ A = {1, 2, 3, 4, 5}
B = {5, 6, 7}
print(A|B) → {1, 2, 3, 4, 5, 6, 7}
→ using union()
print(A.union(B)) → "

2) Set intersection → Intersection of A & B is a set of elements that are common in sets.
→ It is performed using &.
→ eg ⇒ print(A&B) → {5}.
→ Same can be accomplished by using intersection()
print(A.intersection(B)).

3) Set Difference → Diff - of A & B (A-B) is a set of elements that are only in A but not in B, similarly (B-A) set of elements in B but not in A.
* It is performed using - operator / difference()
* eg → print(A-B) → {1, 2, 3, 4}.

→ Set Build-in ():-

1) all()
2) any()
3) len()
4) max()
5) min()
6) sum().

Module = 3

04 : Functions

* () →
  * () is an organised, reusable, self-contained prgm segment that carries out some specific, well defined task.
* * () contained its indented action whnevr it is accessed.
* 3 types of () →
  a) python build-in ().
  b) user defined ()
  c) anonymus () → no () name

* () elemnts → . name.
  . paramtrs → input passing prgm values
  . Returns type. / result type.

eg → def msg():
  print("hello At")
msg().

* pass paramtes → create () →
eg → def msg(naye):
  print("hello" + name)
msg("At")

→ ()s →

a) abs() → absolute value of a no~. The arg may be an int / float.
  Syntx → abs(n).
  eg → int = -20
  print(abs(int)) → 20//

2) all() → iterable → list, tuple, dict
return true.

returns T if all items in an iterable are T, otherwise F.
If iterable obj is empty → returns T.
Syntx → all (iterable)
print value → iterable

eg → L = [0,1,2]
print (all (mylist)) → T.

2) any() → anyone → T.
returns T, if any item in an iterable are T, otherwise F.
If the iterable obj is empty → F.
Syntx → any (iterable)
eg → L = [1,3,4,0]
print (1, any (L)) → T.

L = [0, false]
print (any(L)) → f

3) ascii() → returns a readable version of a obj (str, tuples, list, etc).
This () replace any non ascii char with escape char.
eg → ö → \xf6n
ascii
Syntx → ascii (obj)
eg → a = ā
print (ascii (aC)) → 65

4) bin() → returns binary version of a specified int.
result will always start with prefix ob
Syntx → bin (n)
eg → Ak = 54
print (bin (Ak)) → ob110110
(output ob means start number)

5) bool() → returns boolean value of an specified obj.
obj will always return False unless
returns false: (1) object is empty like [], (), {}
(2) obj is false
(3) obj is 0
(4) obj is none
Syntx → bool (obj)
eg → to) = []
print (bool (to)) → F.

6) chr() → returns char that represent the specified unicode.
Syntx → chr (number).
number → int representing a valid unicode point.
eg → print (chr (97)) → a

7) complex() → returns a complex no. by specifying a real no. & Imag no.
Syntx → complex (real, img)
eg → print (complex (2, -3)
→ 2-3i

8) dict() → creates a dict of array.
s/x → dict (keyword, arg)

9) float()

10) input()

8) hex()

10) int()

11) len() → no of items in an obj.
s/x → len (obj)
l = [1, 2, 3]
print(len(l)) → 3

12) list() =
creates a list obj.
s/x → list (iterable)

13) max() → returns highest value.
s/x → max(arg1, arg2 ...).
max (iterable)
eg → AK = [1, 2, 3, 10]
print(max(AK)) → 10

14) min() →
print (min(AK)) → 1

15) next() →
A = iter(["apple", "banana"])
x = next (A)
print (x)
x = next (A)
print(A). → apple
banana

16) oct() →
oct(int) 0-8
eg → print(oct(10)) → 0o12

17) ord() → unicode char.
s/x → ord (char).
eg → print (ord('5')) → 53.
char→str

18) pow() → pow(x, y, z).
z → optional.
eg → print(pow (4, 2)) → 16

19) print()

20) range()

21) round() → round decimal values.
s/x → round(number [, digits]).
number → required. the no. to be rounded.
digits → optional, the no. of decimals to use when rounding the no., default is 0.
eg → print(10.7) → 11

22) set() → creates set.

23) sorted() → returns sorted list.
s/x → sorted (iterable, key=key, reverse=reverse)
* iterable → required. The sequence to sort list, dict, tuple...
* key → optional, λ() to execute to decide the order
default → none.
* reverse → optional. A boolean.
False → ascending
True → descending.
default → False.
eg → print(sorted(1, 5, 3, 0))
→ [0, 1, 3, 5].

24) str() → convert to str
25) sum() → sum(iterable, start)
optional

26) tuple()
27) type()

```
from math import sqrt, log
"    "   import *
```

→ math ()

1) math.ceil(x) → for rounding.
2) math.comb(n,r) → for combination.
3) " .factorial(x)
4) " .gcd(a,b) → like LCM.
5) " .sqrt(x)
6) " .remainder(x,y)
7) " .power(x,y)

* trignometric ()
8) math.sin(x)

Hyperbolic ()
9) math.sinh(x)

* mathconstnts
10) math.pi()
11) math.tau()

→ Date () =
12) date.today() → print local data print.
13) datatime.date(yr, month, day)

```
import datetime
```

* global variables → out.
* local " → inside a ()

=> Anonymous () = It is a () that is defined without a name that are defined using lambda keyword
Str → lambda arguments : exp.
eg → double = lambda x : x * 2
print (double(5)). → 5*2 = 10