[10, 20, ... 'str'] → list
(10, 10.20) → tuple

03 : Strings, lists & tuples in python

* String operator → = [assessing]
* 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 character out of index range will rise 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])        → ~~[illegible]~~
→ orld
```

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

```
ak [-5 : -2]
orld
```

```
b = "ansarbb"
print(b[2:6:2])
s a r b b
→ s 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 dlt / remove char frm a str, dlting the str entirely is possible using the keyword 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 allowed 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 (+).
* → repeatation → creates new str, concatinating multiple copies of the same str.
[ ] → slice
[:] → rangeslice
gives character from givn index.

[eg] + → a = 'hello'
    b = 'python'.
    c = a+b
    print (c)
* → a = 'hello'
    ~~'python'~~
    a * 2
    → hellohello.
[ ] → a[1]
    → e
[:] → ~~a =~~ [A|k]

0	1	2	3	4
A	n	s	a	r

a = Ansar
a [1 : 3] → ns
→ gives the characters from givn range.

in → returns true if a character exist in the givn str.
    a = 'Hello'
    print ('H' in a) → True.
not in → returns true if a character does not exist in givn 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)

is → 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) ~~[illegible]~~ isalnum() →
Returns True if all characters are alpha numeric → (A-Z, a-z, 0-9)
sple 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 are numeric. (0-9) otherwise false
Syntx → str.isnumeric ()

11) isspace () :-

The is space method returns true if all the characters in string are white spaces

g:- str.isspace ()
str. is space ()

=> list operations :-

(i) 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 → num 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,~~8~~,12

→ 1 2 3 4
1 2 3 12 4

3) count() :- Returns count of how many times object 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']
~~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 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 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 from a tuple.
But deleting a tuple entirely if possible using the using the key word "del".

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

5) Count () :- returns nom 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 ~~and~~ 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
```

⇒ Python dict =

A dict is a collection of indexed changeble & 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 from dict →

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

* Indexing is used with other containing time to access values, dict users 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 () is that it returns none instead of key error if the key is not found.

⇒ Dlt or remove elements from 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()
or dlt

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

* pop item () used to remove & return an arbitary items (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 (31)) → {2:4, 3:9, 5:25}.
del →           del. S [2].        → {3:9 , 5:25}.
remove allitems → 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 return F.

3) len() → returns the length.

4) cmp() → compare 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.
Hwevr the sets itself is mutable you can add / remove items frm it.

* A Set is created placing all elements inside {}.

* Sets can hve any non of elements & they may be of any datatype.

* eg → Set = {1, 2, 3}.
print(Set).

* set () → A str can be parsed to set() as well.

eg → # Set of chr fom 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)       → ~~{1,3,5,2,3,4}~~
                                  → {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 remaines unchanged, but remove () will rise an error in such condition.
* eg →

```
        set = {1,3,4,5,6}
        print(set.discard(4))
        print (set.remove(6))
                 → {1,3,5}
```

union → |
Intrsection → &
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 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 segmnt that carries out some specific, well defined task.
* A () contained its indented action whenevr it is accessed.
* 3 types of () →
  a) python build-in ().
  b) user defined ()
  c) Anonymus () → no () name

* () elemnts → • name.
  • paramtrs → input നമ്മൾ കൊടുക്കുന്ന values
  • Return type. / result type.

```
eg → def msg():
         print ("hello Ak")
     msg().
```

* pass paramter → create () →

```
eg → def msg(name):
         print("hello" + name)
     msg ("AK")
```

→ ()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.
Sytx → all(iterable)
prmtr value → iterable ~~[illegible]~~

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 (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 → $\underbrace{\backslash xf6n}_{ascii}$
S/x → ascii (obj)
eg → C = (ä)
print(ascii (C)) → 65

4) bin () → returns binary version of a specified int.
result will always start with prefix 0b
S/x → bin(n)
eg → Ak = 54
print (bin(Ak)) → 0b110110
(output always start 0b)

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

6) Chr() → returns char that represent the specified unicode.
S/x → 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. & img no.
S/x → complex (real, img)
eg → print (complex (2, -3))
→ 2-3i

8) <u>dict()</u> → Creates a dict of array.
S/x → dict (keyword, arg)

9) <u>float()</u> (9) <u>input()</u>

8) <u>hex()</u> (10) <u>int()</u>

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

12) <u>list()</u> =
creates a list obj.
S/x → list (iterable)

13) <u>max()</u> → returns highest value.
S/x → max(arg1, arg2....). or
max (iterable)
eg → Ak = [1, 2, 3, 10]
print(max(Ak)) → 10

14) <u>min()</u> →
print (min(Ak)) → 1

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

16) <u>oct()</u> →
oct (int) (0-8)
eg → print (oct(10)) → 0o12

17) <u>ord()</u> → unicode char.
S/x → ord (char).
eg → print (ord('5')) → 53.
char→str

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

19) <u>print()</u>
20) <u>range()</u>
21) <u>round()</u> → 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) <u>set()</u> → creates set.

23) <u>Sorted()</u> → returns sorted list.
S/x → Sorted (iterable, key=key, reverse=reverse)
* iterable → required. The sequence to sort list, dict, tuple..
* key → optional, A () 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,k) → for Combination.
3) " .factorial(x)
4) " . gcd (a,b) → like LCM.
5) " . sqrt (n)
6) " . reminder (x,y)
7) " . power(x,y)
* trignometric ()
8) math. sin (x)
Hyperbolic ()
9) math . sinh (x)
* math constnts
10) math.pi()
11) math. tau ()
→ Date () =
12) date.today () → crnt local data print.
13) datetime.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
Stx → lambda arguments : exp.
eg → double = lambda x : x * 2
print (double (5)). → 5*2 = 10