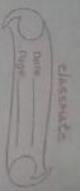


[10, 20, ... 80] - list
(10, 100, 1000) - tuple



03: Strings, lists & tuples in python

* String operator \rightarrow = [accessing]
Slicing \rightarrow Accessing Substring
`print("Hello, world")`

`b = "Hello, world"`
`print(b[0:5])` \rightarrow Hello

String is immutable

\rightarrow Strings in python :-

`a = "Hello world"`
`print(a[0])` \rightarrow ~~error~~

* You can return a range of characters by using the slice syntax, specifying the start index & end index, separated by a colon, to return a part of string \rightarrow slicing.

`b = "ak-programz"`
`print(b[0:10])` \rightarrow ak-programz

* Try to access a character out of index range \rightarrow will raise 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 and last item to the last item.

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

\rightarrow world

`ak[-5:-2]`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671	672	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687	688	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703	704	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735	736	737	738	739	740	741	742	743	744	745	746	747	748	749	750	751	752	753	754	755	756	757	758	759	760	761	762	763	764	765	766	767	768	769	770	771	772	773	774	775	776	777	778	779	780	781	782	783	784	785	786	787	788	789	790	791	792	793	794	795	796	797	798	799	800	801	802	803	804	805	806	807	808	809	810	811	812	813	814	815	816	817	818	819	820	821	822	823	824	825	826	827	828	829	830	831	832	833	834	835	836	837	838	839	840	841	842	843	844	845	846	847	848	849	850	851	852	853	854	855	856	857	858	859	860	861	862	863	864	865	866	867	868	869	870	871	872	873	874	875	876	877	878	879	880	881	882	883	884	885	886	887	888	889	890	891	892	893	894	895	896	897	898	899	900	901	902	903	904	905	906	907	908	909	910	911	912	913	914	915	916	917	918	919	920	921	922	923	924	925	926	927	928	929	930	931	932	933	934	935	936	937	938	939	940	941	942	943	944	945	946	947	948	949	950	951	952	953	954	955	956	957	958	959	960	961	962	963	964	965	966	967	968	969	970	971	972	973	974	975	976	977	978	979	980	981	982	983	984	985	986	987	988	989	990	991	992	993	994	995	996	997	998	999	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	138
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	-----

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

You cannot alter / remove char from a str, altering 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 allow for including special characters into str.

eg → `newline` → `\n`
`backslash` → `\\`
`'` → single slash
`backspace` → `\b`
`horizontal tab` → `\t`
`vertical tab` → `\v`

* String operators :-

+	*	[]	[:]	in, not in
concat	repetition	slice	range/slice	
add values of str (e.g. 'a' + 'b')	creates new str, concatenating multiple copies of the same str.	gives character from given index.		

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

[] → `a[]`

[:] → `a[:]`

`a = 'helloworld'`
`a[1:3]` → 'he'

→ gives the characters from given range. in → returns true if a character exist in the

eg → `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 = 'ansay'`
`print(a.capitalize())`

returning a str where the 1st character is uppercase
 Syntax → `string.capitalize()`

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

Ps- → TIF.

3) Casefold() → returns a str where all the

* Characters are lowercase.

* This method is similar to lower() method.
* But the casefold is strong, more aggressive, meaning that it will find more characters to lowercase & will find more matches when comparing 2 strings & both are converted using casefold method.

4) isalpha() →

returns True if all characters are alpha numeric → (A-Z, a-z, 0-9)
space characters X.
Syntax → str.isalpha()

5) isalpha() →

returns True if all characters are alphabetic (A-Z, a-z)
Syntax → str.isalpha()

6) isdecimal() → 0-9

Syntax → str.isdecimal()

7) isdigit() → returns True if all

characters are digits otherwise False.
Syntax → 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, a-z, 0-9, _).
* A valid identifier cannot start with a non-alphabetic character.

Syntax → str.isidentifier()

9) islower() :-

This method returns True if all characters are in lowercase.
Syntax → str.islower()

10) isnumeric() :-

This method returns True if all characters are numeric (0-9) otherwise False.
Syntax → str.isnumeric()

11) isspace() :-

This is space method returns True if all the characters in string are white spaces.

Syntax → str.isspace()
str.isspace()

⇒ List operations :-

1) append() :-

Syntax → list.append(obj).
appends a post object into the existing list.
obj → object to be appended into the list.

list

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

2) insert() :- append() only works for addition of elements at the end of list.

for addition of element at desired position, insert() is used.
Unlike append() which takes only 1 argument, insert() requires 2 args (pos, value) pos → position.

Syntax → 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(L)
L.insert(3, 12)
print(L)

→ 1 2 3 4
1 2 3 12 4

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

classmate

41

eg = L = ['123', 'AK', '9', 'AK', 'c', 'AK']
print(L.count('AK'))

1) list extend() :- It appends the contents of sequence to L.

L.extend(seq)
eg → L = ['pig', 'che', 'bro']
L2 = list(range(5))
L.extend(L2) → pig, che, bro, 0, 1, 2, 3, 4

→ Tuple() :-

* Basic Tuple operations :-

1) length :-

eg → th = ("apple", "banana", "cherry")
print(len(th))

2) concatenation :-

You can join + operator to combine 2 tuples → .C.
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

classmate

2) Deleting a tuple :-

The elements of a tuple cannot be changed once it has been assigned that also means you cannot remove/delete an item from a tuple.

But deleting a tuple entirely is possible using the del keyword.

eg - `AK = ("programz", "bk")`
~~`del(AK)`~~
`del(AK)`

5) Count() :- returning how many times a

specified value occurs in a tuple.

eg - `t = tuple.count(value)`
`print(t.count(1))` → 2

6) index() :- binds the occurrence

of the specified values. It raises an exception if the value is not found.

eg - `t = tuple.index(value)`
`print(t.index(8))` → 3

⇒ Python dict =

A dict is a collection of unordered ~~key~~ key: value pairs.

* In a dict, keys ~~must~~ must be unique & stored in an unordered manner.

* eg - `AK = {'ab': 15, 'cd': 20}`
`print(AK)`

* Accessing elements from dict ->

~~`AK = {'name': 'John', 'age': 20, 'class': 'BSc CS'}`~~
`AK = {'name': 'John', 'age': 20, 'class': 'BSc CS'}`
`print(AK.get('name'))`

* indexing is used with other containing true to access values, dict uses keys.

* key can be used either inside `[]` as indexing or the get method.

eg - `a = {'name': 'AK', 'class': 'BSc CS'}`
`print(a['name'])` → AK.

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

⇒ Dict as remove elements from dict :-

eg - `AK = {'1': 1, '2': 4, '3': 4, '4': 2}`
`print(AK.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 arbitrary 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) → print(S.pop(4)) → $\{1:1, 2:4, 3:9, 5:25\}$
pop item() → print(S.pop item(4)) → $\{2:4, 3:9, 5:25\}$

del S[2] → $\{3:9, 5:25\}$

del S → del S → $\{\}$
del S → del S → $\{\}$

→ Build-in () =

1) all() → returns true if all keys of dict are true (or) if 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.
(Example)

5) sorted() → returns 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 & unordered 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 from it.

* A set is created placing all elements inside $\{\}$.

* Sets can have any no. of elements & they may be of any data type.

eg → set = $\{1, 2, 3\}$
print(set)

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

eg → set of chr from str.
 $AK = \text{set('AK programming')}$
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 / others 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, 3, 4}

→ Removing elements from set =

* A particular item can be removed from set using discard() if remove()

* The only diff b/w 2 is that while using discard(), if the item does not exist in the set, it remains unchanged, but remove() will give 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) 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

* eg → print(A - B) → {1, 2, 3, 4}

→ Set build-in () :-

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

Q4 : Functions

Module = 3

* () →

* () is an organised, reusable, self-contained program segment that carries out some specific, well defined task.

* () contained its indented action whenever it is called.

* 3 types of () →

- a) python build-in ().
- b) user defined ()
- c) anonymous () → no () name

* () elements → name.

- parameters → input receiving
- return type / result type.

eg → def msg ():

print ("hello ~~At~~ ")

msg (),

* param parameters → create () →

eg → def msg (name):

print ("hello " + name)

msg ("At")

→ () →

a) abs () → absolute value of a num. type

any way be an int / float.

eg → int → abs (4).

eg → int = -20

print (abs (int)) → 20

2) all() → returns → list, tuple, dict pattern true.

returns T if all items in an iterable are T, otherwise F.

If iterable obj is empty → returns T.
Syntax → all(iterable)

Prints value → iterable

eg → L = [0, 1, 2]

print(all(mylist)) → T.

2) any()

→ returns → T.

returns T, if any item in an iterable are T, otherwise F.

If +ve iterables obj is empty → F.

Syntax → 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, tuple, list, etc).

This () replace any non ascii char with escape char.

eg → 0 → \x00

eg → 0 → \x00

eg → 0 → \x00

eg → 0 → \x00

print(ascii(0)) → 65

4) bin() → returns binary version of a specified int.

returns int. specified with prefix 0b

Syntax → bin(n)

eg → At = 54

print(bin(At)) → 0b110110
(output always start with 0b)

5) bool() → returns boolean value of an specified obj.

obj will always return True/False

returns (1) object is empty like [], {}, {}
(2) obj is false.
(3) obj is 0
(4) obj is none

Syntax → bool(obj)

eg → test = []

print(bool(test)) → F.

6) chr() → returns char that represent the specified unicode.

Syntax → chr(unicode)

number → int representing a valid unicode point

eg → print(chr(97)) → a

7) complex() → returns a complex no. by specifying a real no. by imaginary no.

Syntax → complex(real, imag)

eg → print(complex(2, -3))

→ 2-3i

8) dict() → creates a dict of any
\$hr. → dict(keyword, any)

9) float() (9) input (5)

8) hex()

(10) int (5)

11) len() → no. of items in an obj.
\$hr. → len(obj)

l = [1, 2, 3]

print(len(l)) → 3

12) list() = creates a list obj.

\$hr. → list(iterable)

13) max() → returns highest value.
\$hr. → max(arg1, arg2, ...)

max(iterable)

eg → arr = [1, 2, 3, 10]

print(max(arr)) → 10

14) min() → print(min(arr)) → 1

15) next() →

A = iter(["apple", "banana"])

x = next(A)

print(x)

x = next(A)

print(A)

→ apple
→ banana

16) oct() →

oct(int)

eg → print(oct(10)) → 012

17) ord() → unicode char.
\$hr. → ord(char)

eg → print(ord('5')) → 53

18) pow() → pow(x, y, z)

eg → print(pow(4, 2)) → 16

19) print()

20) round() → round decimal values.
\$hr. → round(number [, digits])

number → required, the no. to be rounded.

digits → optional, the no. of

decimal to use when rounding

eg → print(round(10.7)) → 11

21) set() → creates set.

22) sorted() → returns sorted list.

\$hr. → sorted(iterable, key=key, reverse=reverse)

* iterable → required, the sequence to

sort list, dict, tuple.

key → optional, A() to attribute to

decide the order

reverse → optional, A boolean.

False → ascending.

True → descending.

eg → print(sorted([1, 5, 3, 0]))

→ [0, 1, 3, 5]

24) $\delta n()$ \rightarrow convert to δn
25) $\delta n()$ \rightarrow δn (iteration, δn).

26) $\delta n()$
27) $\delta n()$

from math import δn , log
import *

\rightarrow math ()

1) math.~~ceil~~(x) \rightarrow for rounding.

2) math.comb(n,r) \rightarrow for combination.

3) " . factorial(x)

4) " . gcd(a,b) \rightarrow like lcm.

5) " . log(n)

6) " . remainder(x,y)

7) " . power(x,y)

8) math.sin(x)

9) math.sinh(x)

10) math.pi()

11) math.tan(x)

12) Date () =

13) date.today() \rightarrow print local date print.

14) datetime.date(yr, month, day)

import datetime

global variables \rightarrow out.
local " \rightarrow inside a ()

\Rightarrow Anonymous () = it is a () that is

defined without a name that are

defined using lambda keyword

eg \rightarrow lambda arguments: exp.
print (lambda (x): x * 2) \rightarrow 5 * 2 = 10