List and Tuples

→ These are called compound data types and are one of the key types of data structures in Python.

* Tuples :- Tuples are an ordered sequence

- Here is a Tuple "Ratings"
- Tuples are written as comma-seperated elements within parentheses.

Ratings = (10, 9, 6, 5, 10, 8, 9, 6, 2)

eg. tuple 1 = ('disco', 10, 1.2)

type (tuple 1) = tuple

Each element in Tuple can be accessed via index.

Tuple 1 = ("disco", 10, 1.2)

-3	0	disco
-2	1	10
-1	2	1.2

↓ negative index

concanted tuple :

tuple 2 = tuple 1 + ("hard rock", 10)

("disco", 10, 1.2, "hard rock", 10)

0	1	2	3	4

We can slice tuple,

For First 3 elements.

tuple 2 [0:3] : ('disco', 10, 1.2)

Last index is more than one index we want.

tuple 2 [3:5] : ("hard rock", 10)

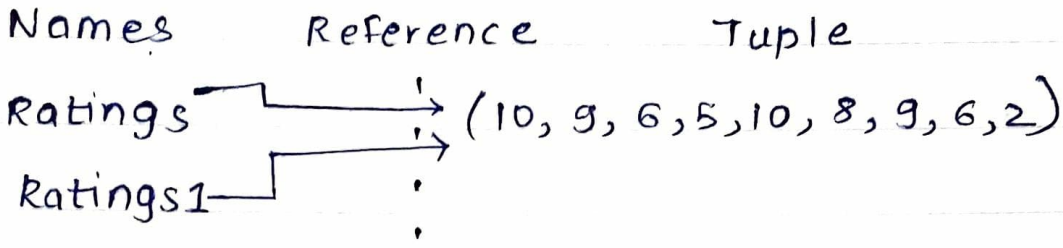
↓ one larger than size of tuple

len command ⇒ Length of the tuple

* Tuples are immutable, which means we can't change them.

Ratings: (10, 9, 6, 5, 10, 8, 9, 6, 2)

Ratings1 = Ratings



we can'nt change element in the ratings rating1 will not be effected by a change in rating.

We assign different tuple to variable,

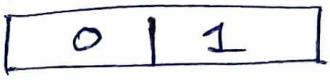
Ratings = (2, 10, 1)

* Ratings Sorted = sorted (Ratings)

Tuple can contain other tuple as well as other complex data types → nesting.

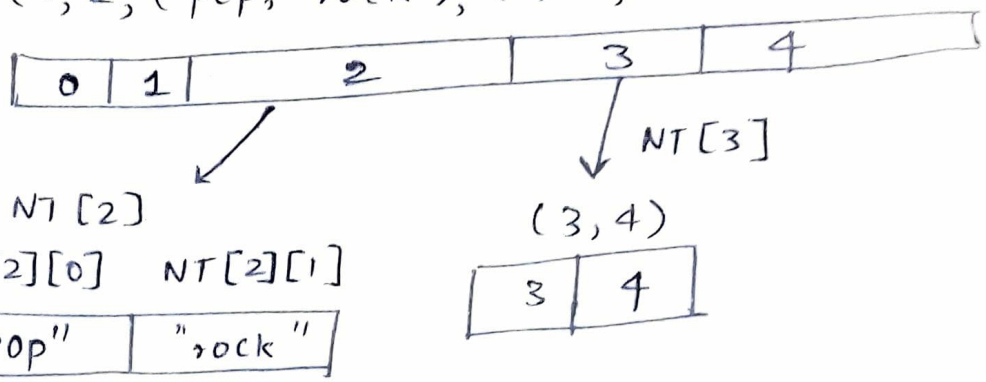
NT = (1, 2, ("pop", "rock"), (3, 4), ("disco", (1, 2)))

NT[2]: ("pop", "rock")[1] = "rock"



NT[2][1] = "rock"

NT = (1, 2, ("pop", "rock"), (3,4), ("disco", (1,2)))



* Lists :- Lists are also ordered sequences

- Here is a list "L"
- A List is represented with square brackets.

L = ["Michael Jackson", 10.1, 1982]

- List mutable

-3	0	"Michael Jackson"
-2	1	10.1
-1	2	1982

index

L[0] : "Michael Jackson"

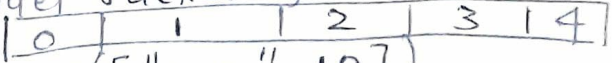
L = ["Michael Jackson", 10.1, 1982, "MJ", 1]

L[3:5] : ["MJ", 1]

→ one larger than length of list

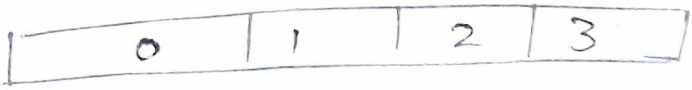
L.extend(["pop", 10])

L = ["Michael Jackson", 10.1, 1982, "pop", 10]



L.append(["pop", 10])

L = ["Michael Jackson", 10.1, 1982, ["pop", 10]]



A = ["hard rock", 10, 1.2]

del(A[0]) del(A[1])

A : [10, 1.2] A : ["hard rock", 1.2]

* Convert string to List

"hard rock".split()

["hard", "rock"]

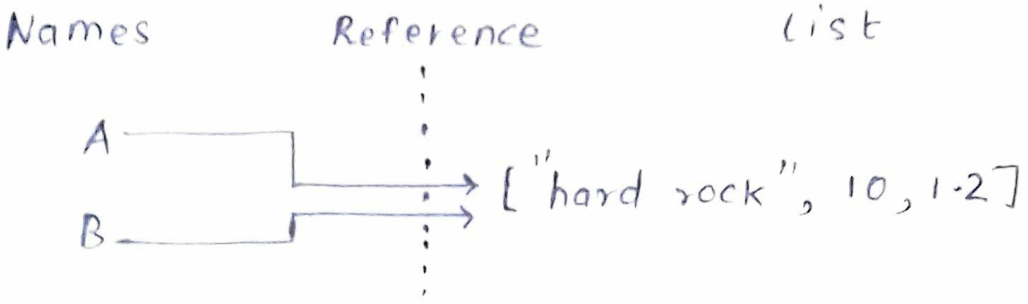
"A, B, C, D".split(",") — delimiter

= ["A", "B", "C", "D"]

Lists : Aliasing

A = ["hard rock", 10, 1.2]

B = A



B[0] = "hard rock"

A[0] = "banang"

if we change List A list B also changes.

clone :- A = ["hard rock", 10, 1.2]

B = A[:]

A ⟶ ["hard", 10, 1.2]

B ⟶ ["hard", 10, 1.2]

if we change A, B will not change.

help(A)

① first element A[0]

② A[2] : [4]

③ [4]

④ ('A', 'B', 'C', 'D')

⑤ append ; merges two lists

⑥ list : mutable (True)

⑦ ["hard R", 1.2]

⑧ Refered list A

* Dictionaries :- Dictionaries are a type of collection in python.

List

0	Elem 1
1	Elem 2
2	Elem 3
...	...

Dictionary

key 1	value 1
key 1	value 2
key 2	value 3

Key is analogous to the index.

They are like addresses, but they don't have to be integers. They are usual characters.

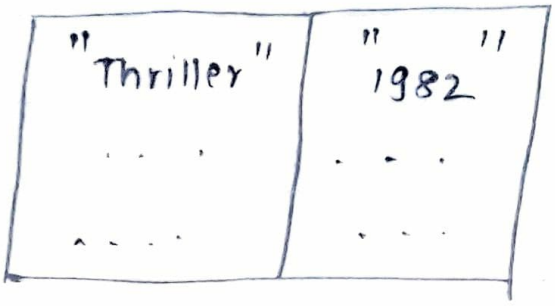
To create dictionary :-

- Dictionaries are denoted with curly Brackets { }
- The keys have to be immutable and unique

{"key1": 1 "key2": 2 "key3": [3,3,3] "key4": (4,4,4) ('key 5'): 5 }

- The values can be immutable, mutable, and duplicates.
- Each key and value pair is separated by a comma

DICT =



DICT ["Thriller"]: 1982

Add new element :- DICT ['Graduation'] = '2007'

Delete : del (DICT ['Thriller'])

To verify in command :-

'The Bodyguard' in DICT ⟶ True or False

DICT.keys() = ["...", "...", "..."]

DICT.values() = [. . . . .]

Sets :- type of collection

→ This means that like lists and tuples you can input different Python types.

- Unlike lists and tuples they are unordered

→ This means sets do not need record element position.

- sets only have unique elements.

→ This means there is only one of a particular element in a set

* Creating a set

Set 1 = { "pop", "rock", "soul", hard rock", ... , "disco" }

When actual set created duplicate item is not will not be present

* convert list → set() ⟹ By using function set type casting

e input → list to the function set

The result will be a list converted to a set.

Eg. album_list = ["Michael Jackson", "Thriller", "Thriller", 1982]

album_set = set (album_list)

album_set = {'Michael Jackson', 'Thriller', 1982}

* set operations.

→ A venn diagram is a tool that uses shapes usually to represent sets.

A = {"Thriller", "Back in Black", "AC/DC"}

A.add ("NSYNC")

A : {"AC/DC", "Back in Black", "NSYNC", "Thriller"}

A.remove ("NSYNC")

A : {"AC/DC", "Back in Black", "Thriller"}

→ If an element is in the set using the in command as follows :-

"AC/DC" in A

True

"Who" in A

False

Mathematical set operations

album-set-1 = {"AC/DC", Back in Black", "Thriller"}

album-set-2 = {"AC/DC", "Back in Black",
"The Dark side of the Moon"}

intersection : album-set-1 & album-set-2

union : album-set-1.union (album-set-2)

if it a subset :-

album-set-3.issubSet (album-set1)

True

* conditions and Branching

comparison operations compares some value or operand. Based on cond^n they produce boolean

a=6, a=7 } False

a 6, a=6 } True

i>5, i=6 } True

i>=5, i=5 } True

i!=6 -> True

False

Branching allows us to run different statements for a different input

Pseudo statement

if (age>18):

print ("you can enter")

print ("move on")

else statemet

```
if (age>18):
    print ("you can enter")
else :
    print ("go see meat loaf")
print ("move on")
```

* elif statement

```
if (age>18)
    print ("you can enter")
elif (age==18)
    print ("go see pink floyd")
else:
    print ("go see meal Loaf")
print("move on")
```

*LOGIC operators :-

Boolean

Logic operators

Take boolean values and produce different Boolean values

1. True — not

not(True) = False

False — not

not (false) = True

```
album_year = 1990
if (album_year < 1980) or (album_year > 1989):
    print("The Album was made in the 70's or 90's)
else:
    print("The album was made in the 1980's")

album_year = 1983
if (album_year > 1979) and (album_year < 1990):
    print("This album was made in the 80's)
```