(12)

* Lists → collection of data → exactly what arrays are in C++ and Java (very similar use case) → but heterogeneous type → not immutable

creating lists

```
a = [1, 2, 3]
type(a)
```

a = [] → empty list

```
a1 = list() → empty list
a2 = list([1, 2, 3])
   or
a2 = list(a)
```

↳ can be used to make copies of lists

```
a4 = [0 for i in range(10)]
a5 = [i for i in range(10)]
a6 = [i*i for i in range(10)]
a7 = [1, 2, "as", True]
```

element ↑ (loop to run) → for each iteration, add element

↓

lists can have different datatypes, unlike arrays in C++/Java

accessing → both forms of indexing

slicing normal way

len(a)

↳ length of list can be found

fast iteration possible

* taking input as lists

↓

default input in python is as a string, and if a user doesn't enter after each entry, then 1 2 3 4 will come in as a string and will not be stored in a list as desired

↓

so we need a way/ method to input list efficiently

(split on all whitespaces)

↑

str = " a - b - c - d " → can use split()

** str.split("-") → splits the string about "-" character

** str.strip() → removes white space on both ends of string

* we first strip the string input of extra white space. then split it about space character, which gives us a list of character integers, which need to be converted to int and added to new list. that can be done using fast iteration creation of a string.

inp = input()

* a = [int(x) for x in input.strip().split("-")]

↓ ↓

store x for each iteration of loop

one line code to take input as int list

↓

or

↓

n = input()

l = [int(i) for i in n.split()]

(14) # add elements to list

```
l = [1, "ac", 3]
l.append(2)  → will always insert at end
l.insert(1, 23)
```

↓ inserts 23 at this index (1) and moves other elements to the right

```
l2 = [2, 3, 4]
l.extend(l2)
```

↓ adds completely separate list to the current list (to the end)

↓ inplace

deleting elements in list

l.pop() → last element removed & returned without argument

l.pop(2) → pops element at index 2

↓ all elements to the right move to the left

don't return the element

l.remove(3) → will remove the first occurrence of element 3

del l[3] → delete element at index 3

↓ can given slice delete

↓ if assigned to another variable, it maintains its value

list + integer → not valid

list + list → valid

↳ concatenation through list

l = [1, 2, 3]

l + l → [1, 2, 3, 1, 2, 3]

list * 2 → [1, 2, 3, 1, 2, 3] ⟹ multiplication with int is fine

l.sort()

l.count(1) → frequency of element in list

l.index(5)

↳ returns index of element in list

l.reverse() → list reversal

7 in l → returns bool based on presence of element in list

max(list), → returns max element of list

similarly min(list) can also be used.

* print([i.lower() for i in 'HELLO'])

↓

print list, elements of which satisfy the condition to gain access

(16) * Bubble sort

↓

starts comparing from 0th index to ith index and places largest element to the right most

↓

n scans.

↓

$O(n^2)$ complexity.

```
l = [int(x) for x in input().strip().split()]
n = len(l).
for i in range (n-1):
    for j in range ( 0, n-1-i ).
        if l[j] > l[j+1]:
            t = l[j]
            l[j] = l[j+1]
            l[j+1] = t
print(l).
```

* selection sort

↓

find min and put it at start of array

```
l = [int(x) for x in input().strip().split()]
n = len(l)
m = 0.
for i in range (n):
    m = i
    for j in range (i, n):
```

```
        if l[m] > l[j]:
            m = j
        temp = l[m]
        l[m] = l[i]
        l[i] = temp
print(l).
```

→ sum of indices less than i = sum of indices greater than i

* Equilibrium index of array

```
n = int(input())
l = [int(x) for x in input().strip().split(" ")]

left = list()
right = list()
left.append(l[0])
right.append(l[-1])

for i in range(1, n):
    left.append(left[i-1] + l[i])

j = 1
for i in range(n-2, 0, -1):
    right.append(right[j-1] + l[i])
    j += 1
right.append(right[j-1] + l[0])
j = n-1
flag = 0
for i in range(n):
    if left[i] == right[i]:
        print(i)
        flag = 1
        break
    else:
        j = j-1
```

(18) if flag >=0:
print(-1).

dictionary → curly
tuple → parenthesis
list → [] → square

Dictionaries : similar to map.

↓

(key, value pairs)

↓

the role of index is played by user defined key.

d = { } → creates a dictionary

d[23] = 34

d["str"] = 345

d → {23: 34, 'str': 345}

d = {23: 34, "str": 345}

↓

another way of creating a dictionary

d = {key: value, key: value, key: value, ...}

d[23] = 345 → two possibilities

↓ update if already present

↳ create new, if not

* keys of dictionary are supposed to be immutable

↓

string, int, float, tuple ✓

↓

list, dictionary ✗

fast iteration in d.

→ iterates over keys of dictionary

```
for i in d:
    print (i, end = ":")
    print (d[i]).
```

for key, value in dict.items():
print ("{} : {}".format(key, value))

delete in dictory

del d[23]

accessing non available key → key error.

common functions.

d1 = {}
d2 = {}

d1 == d2 → comparing 2 dictionaries.

↓

when all are same (keys and value pairs, only then is true returned, else false)

len(d1).

d1.clear()

d2.keys() → all keys present in dictionary

d2.values() → all values present in dictionary

23 in d2 → present key in dictionary, bool

↓

check to place before accessing key in dictionary

d2.items() → key, value pairs.

d2.update()