(6)

* Boolean.

a = True → (cap T, other small)
→ keywords for Bool values

type (a) → bool

Relational operators

x = 10
y = 20
x > y → returns a boolean
x >= y

Logical operators : and (&&) → used in python
or (||)

conditions → similar to C++ / Java
↓
basic syntax difference
↓
indentation is compulsory in python (all constructs)
↓
causes error in program
↓
(brackets can be used for conditions optional)

```
if a > b :
    print (a)
elif a == b :
    print (a)
    print (b)
else :
    print (a)
```

```
if True or True :
    if false and True or False :
        print ("A")
    elif false and false or True and True :
        print ("B")
    else:
        print ("C")
```

(evaluated left to right)

→ b.

* Loops

while (condition to follow) :
    proper indentation for loop code

→ we can break from loops using break

* range (1, 10) → Range starts from 0 by default
→ (produces numbers for marked range)
(returns iterable object containing int)

for i in range (1, 10)
↓
runs i in range 1 to 10, i.e. it will start i from 1 and end it at 10. (10 not included)
↓
start is inclusive, whereas end is not inclusive.

* range can have 3 arguments : (i) start value (0)
(ii) end value
(iii) step size (1).

(only allowed for iterables)
↑
fast iteration → only faster to write, efficiency diff nahi hai

(all iterable objects)

s = 'abcd'
for c in s : → c assumes the value of each character in s and accordingly performs the loop till all elements are not exhausted.

better to operate on copy()
↑
(not true for lists, etc.)
↑
any changes do not change looping condition

↓
doable in tuples as well.

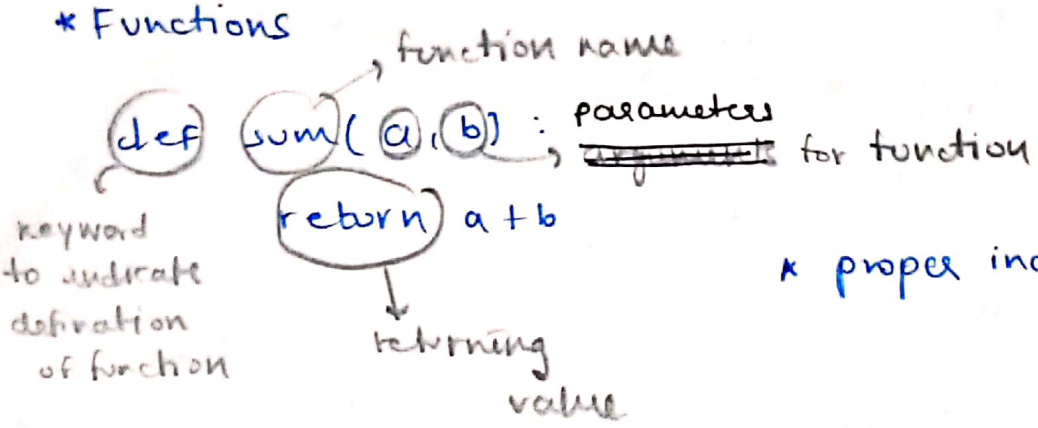
works for its initial state → (executes only initial value times)

a = 3
for i in range (a) :
    print (a, end = " ")
    a += 1

3 4 5

10

* Functions



* proper indentation

Default arguments to a function

↳ need to go from right to left, and that is also how values are assigned incase more values are sent

```
def power ( x , y = 2) :
    return x ** y
```

↳ function will work with even 1 argument

```
def sum ( a = 12, b = 1, c = 2) :
    return a+b+c

print (sum ( 1, 2, 3))
print(sum ( 1, 2) )  ↳ goes to a, b
print (sum (10))  ↳ goes to a
sum ( b = 10)
```

↓

can specify key value pairs

↓

named parameters

variable number of inputs to a function

→ to pack as a tuple

```
def sum(a, b, *more):
    print(more)
    print(type(more))  -> tuple
    return 10.
```

(**kwargs) → (to pack as dictionary)

sum(10, 12, 13, 14)

a b go to the tuple → if no extra variables are given, tuple remains empty.

```
def sum_var(a, b, *more):
    ans = a + b.
    for i in more:
        ans += i
    return ans.
```

sum_var(10, 12, 2, 3, 45, 6, 6)

more than one return value

```
def dummy(a, b):
    return a+b, a-b
```

sum1, diff = dummy(10, 12)

↓ value go into new variables in order

```
a = dummy(22, 10)
print(a)
```

↓ tuple only.

or also a = dummy(22, 10)[0]

↓ another possibility