Data Types & variable.

convert C to f.

```
celcius = int(input('Enter temper'))
frenhite = (celcius * 9/5) + 32
print('celcius', celcius, '=', 'frenhit', fren)
```

Loops.

While loop.

```
number = int(input('enter a no'))
I = 1
While i<11:
    print(number * i) => print(number, '*', i, '=', no*i)
    i+=1

x = 1
While x<3:
    print(x)
else:
    print('limit cross').
```

```
count = 1
while guess != jackpot
```

→ # Guessing game.

```
import random
jackpot = random.randint(1,100).
Guess = int(input('guess kro'))
    if guess < jackpot:
        print('galat ! guess higher')
    else:
        print('galat ! guess lower').
    Guess = int(input('Guess kro')).
                                counter +=1
else:
    print('correct guess').
    print(counter)
```

For loop. Basically use range.

```
for i in range(1, 11, 2):
    print(i)
```

→

```
curr_pop = 10000
for i in range(10, 0, -1):
    print(i, curr_pop)
    curr_pop = curr_pop - 0.1 * curr_pop
```

→

```
n = int(input('enter n'))
result = 0
fact = 1
for i in range(1, n+1):
    fact = fact * i
    result = result + i/fact
print result
```

→ Nested Loops:

```
for i in range(1,5):
    for j in range(1,5):
        print(i,j):
```

i) Pattern:

```
Rows = int(input('enter no of rows'))
    for i in range(1, rows+1):
        for j in range(1, i+1):
            print('*',),
        print().
```

ii)

```
Rows = int(input('Enter number of rows')).
    for i in range(1,5):  -> (1, Rows+1)
        for j in range(1, i+1)
        print('*') -> , end=' ').
    print().
```

output:

iii)

```
rows = int(input('eenter no')
    for i in range(1, rows+1):
        for j in range(1, i+1):
        for k in range(i-1, 0, -1).
        print(j, end=' ')
        Print().
```

2)

```
rows= int(input('enter rows'))
for i in range(1, rows+1):
    for j in range(1, i+1):
        print(j, end = ' ')
    for k in range(i-1, 0, -1):
        print(k, end = ' ')
    print( ) :
```

output:

```
1
121
12321
1234321.
```

→ Break.

```
for i in range(1,10):
    if i == 5:
        break
    print(i).
```

→

```
lower = int(input('enter lower range'))
upper= int(input('enter upper range'))
for i in range(lower, upper+1):
    for j in range(2, i):
        if i%j == 0:
            break.
    else:
        print(i).
```

→ continue.

```
for i in range(1,10):
    if i == 5:
        continue
    print(i)
```

→ Reverse index.

```
print[::-1]
```

→ Find.

```
'my name is'.find('is')
```

→ 'Initi123'.isalnum() → True.

.isdigit()

'name1'.isidentifier() → True

→ 'my name is'.spilt() or (spilt('i').

→ '--'.join(['hi', 'is', 'a'])

.replace('initis')

Function. parameter.

→ def ii_even(num)":
"""" This function give a even-no"):
if num%2 == 0:
return 'even'
else:
return 'odd'.
for i in range(1,11):
x = is_even(i) → argument.
print(x).

→ 2 Point of views.

→ Default Argument.
def power(a=1, b=1):
return a**b
print(power) or power().

→ positional argument.
power(2,3) = 8.

→ keyword argument.
power(b=3, a=2) = 8

→ Args
to pass.
To pass a variable of non-keyword argument.
def multiply(*arg):
product = 1.
for i in args.
product = product * i
return product.
multiply(1,2,3,6,8,10,12).

is_even.__doc__

↳ documentation.

→ KWargs: (key, value pair).
to pass any no of keyword argument.

```
def display(**kwargs):
    for (key,value) in kwargs.items():
        print(key, "-->", value).
display(india='delhi', pak='islan').
```

Rule:
normal, *args, kwarg **

python automatically convert kwagr into dictionary.

→ Without using return.
python will give you "None" in place of this but print can't work.

→ local vs global variable.
within function → local variable

```
def g(y):
    print(x)
    print(x+1)

x = 5
g(x)
print(x).
```

→ Functions are 1st class citizen.

```
def square(num):
    return num**2
    type(square).
    id(square)
x = square
print(x)
```

→ # storing:

```
L = [1,2,3, square].
L[-](3).
```

→

```
def f():
    def x(a,b):
        retuon a+b
    return x
val = f()(3,3)          ⇒ output 6
print(val).
```

→ Lambda, function.

ایسی فنکشن جس کا نام نہیں ہوتا - جو کچھ return نہیں کرتا -
جو ایک لائن میں ہی ہوتے اور (Not reusability) -

used by higher order funtin.

```
Lambda a,b : a+b.
```

```
a = lambda x : x**2
a(2)                output= 4.
```

```
a = lambda x,y : x+y.
a(5,2)              output = 7.
```

→

```
a = lambda s:'a' in s
a ('hello')   →  false.
```

→

```
a = lambda x: 'even' if x % 2 == 0 else 'odd'
a(6)     ⇒ 'even'
a(5)     ⇒ 'odd'
```

⇒ Higher order function: جو input میں کسی کو receive کرے۔

⇒ Map. کسی بھی [] کے اوپر تبدیلی کرنی ہے۔

```
list(map(lambda x : x**2, [1, 2, 3, 4]))

→ list(map(lambda x : 'even' if x%2==0 else 'odd', L)))

→ list(map(lambda users: users['gender'], users)).
```

⇒ Filter

```
L= [3, 4, 5, 7]
→ list(filter(lambda x : x>5, L))

→ fruit [ 'apple', 'bana' ].
  list(filter(lambda x : x.startswith('a'), fruits)).
```

⇒ Reduce

```
import functools.
functools.reduce(lambda x,y:x+y, [1,2,3,4,5]).
```

اپنے data کو private کر کے attributes
سے getter اور setter سے اختیاط کر سکتے ہیں۔

OOPS Encapsulation

constructer (special power).

→ class person:
    def __init__(self, name): ہر obj میں اسکی value مختلف ہو سکتی ہے
        self.name = name (instance variable).

→ private and Public.
    self.__name = name.
obj = person()
obj._person__name = 'hehe'.
python میں ہم چیز کو private پر مانند نہیں کر سکتے

→ def get_balance(self):
        return self.__balance.   ہم balance کو سیٹ بھی
def set_balance(self, value):   کر سکتے ہیں اور get سے
        self.__balance = value.   جو کچھ بھی سکتے ہیں

→ self.pin
    ↳ obj → instance variable.   جو کہ obj سے ہوتا ہے۔
ورنہ class کا static ویری ایبل ہوتا ہے۔ یہ method کے اوپر ہوتا ہے۔
a staticmethod کے لیے obj بنانے کی ضرورت نہیں ہے صرف کلاس use کر کے۔
utility func ہوتے ہیں۔

→ Aggregation. (has a Releation).
    one class owns the other class.
    class customer:
        def __init__(self, name, gender, address):
            self.name = name
            self.gender = gender
            self.Address = adress.
    class Address:
        def __init__(self, city, code, state):
            self.city = city
            self.Pode = code
            self.state = state.

Polymorphism → Basically mean different forms. They have three faces give below.

① ... method ... overriding.

② Method overloading

... methods ... both have diff behaviour.

... clear ... code ... (غائب)

↳ Python they not work Java worked.

③ operator overloading

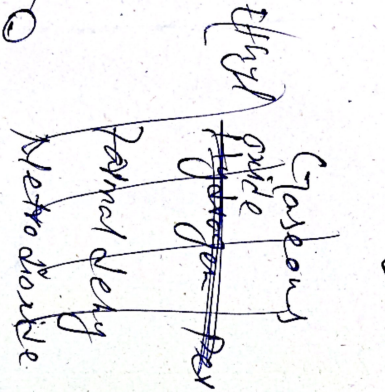
'hello' + 'word'

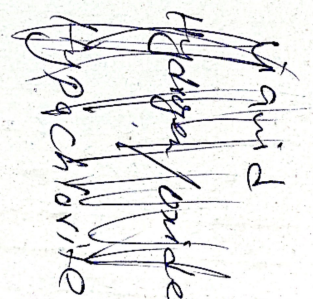
↳ concatination.

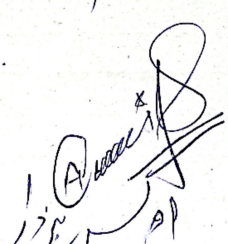
[1,2,3] + [1,3,4]

↳ merge.

... diff ...







```
def print_address(self):
    print(self.address.city, self.code, self.state).

addr = Address('8480e', 122201, 'haryani').
cust = Customer('Awais', 'male', addr)
cust.print_address().
```

→ inheritance. ... create a parent and child class.
code Reusability increase.
... execute ... constructor ... child class ...
... parent ... child ... error ... execute
child can't access private attribute.

→ method overriding.
... child ... same child ... parent

→ Super keyword.
... parent ...
Super.__init__(price, brand, comra).

→ outside class calling. (not work).
... access ... method ...

→ parent can't access child class.