Abdul:

Variable length

print()

print(10)

print(10,12.5)

print(10,12.5,"hello",5,True)

* print function can take variable length arguments.

def fun(\*args):

1. \*args🡪 is used for variable length arguments
2. Tuple is created for variable length arguments

def fun(\*args):

    print(args)

    # print(type(args))

    # #<class 'tuple'>

    for i in args:

        print(i,end=" ")

fun(5)

#(5,)

fun(5,10)

#(5,10)

fun(5,10,15)

# (5,10,15)

fun(5,1.25,"hello",15)

        #5 1.25 hello 15

Prog 2:

def fun(\*args):

    for x in args:

        if type(x)==int:

            print(x)

            #10 15

fun(10,12.5,"hello",True,3+4j,15)

1. fun(a,b,\*args)

a and b are positional arguments only

def fun(a,b,\*args):

    print(a,b,args)

fun(10,20,30)

        #10 20 (30,)

fun(10,20,30,40,50)

    #     10 20 (30, 40, 50)

fun(a=10,b=20,30,40,50)

  SyntaxError: positional argument follows keyword argument

1. fun(\*args,a,b)

a,b are keyword arguments only

prog1:

def fun(\*args,a,b):

    print(a,b,args)

fun(10,20,30,40,50)

# TypeError: fun() missing 2 required keyword-only arguments: 'a' and 'b'

Prog 2:

def fun(\*args,a,b):

    print(a,b,args)

fun(10,20,30,a=40,b=50)

# 40 50 (10, 20, 30)

1. fun(\*L1)🡪 unpacking the list elements

def fun(\*args):

print(args,len(args))

L1=[10,20,30]

fun(L1)

#([10, 20, 30],) 1

L1=[10,20,30]

fun(\*L1)

# (10, 20, 30) 3

A screen shot of a computer

AI-generated content may be incorrect.

Variable length keyword arguments:

#Variable length arguments--> Keyword

def fun(\*\*kwargs):

    print(kwargs)

1. \*\*kwargs is used for variable length keyword arguments
2. Dictonary is created for kwargs

# Variable length arguments--> Keyword

Case 1:

def fun(\*\*kwargs):

print(kwargs)

print(type(kwargs))

# {'a': 5, 'b': 10, 'c': 15}

# <class 'dict'>

fun(a=5,b=10,c=15)

prog2:

def fun(\*\*kwargs):

    for item in kwargs.items():

        if item[0]=='b':

            print(item[1])

            #10

fun(a=5,b=10,c=15)

#dict\_items([('a', 5), ('b', 10), ('c', 15)])

1. Fun(\*\*kwargs,a,b) 🡪 arguments not passed after keyword args

def fun(\*\*kwargs,a,b):

print(kwargs)

fun(a=5,b=10,c=15)

# SyntaxError: invalid syntax

1. Fun(a,b,\*\*kwargs)

a, b are positional/keyword arguments

def fun(a,b,\*\*kwargs):

    print(kwargs)

    # print(a,b,kwargs)

fun(a=5,b=10)

   #{}

fun(a=5,b=10,c=15)

#{'c': 15}

fun(1,2,c=3)

    #print(a,b,kwargs)    #1 2 {'c': 3}

#works for both keyword and positional

1. Fun(\*args,\*\*kwargs)

Always “\*args” will be first

1. Fun(\*\*args,a,b,,\*\*kwargs)

a,b are always keyword not positional.

def fun(\*args,a,b,\*\*kwargs):

    print(a,b,kwargs)

    print(args)

fun(1,2,3,4,x=5,y=10)

#TypeError: fun() missing 2 required keyword-only arguments: 'a' and 'b'

fun(1,2,a=3,b=4,x=5,y=10)

  print(args)  --> (1,2)

  print(a,b,kwargs) -- 3 4 {'x': 5, 'y': 10}

A person pointing at a white board

AI-generated content may be incorrect.

Multiple Returns:

def fun(a,b,c):

    sum=a+b+c

    prod=a\*b\*c

    return sum,prod

print(fun(5,10,15))

    #(30,750)

print(type(fun(5,10,15)))

    # <class 'tuple'>

Prog 2:

def result(marks1,marks2,marks3):

    total=marks1+marks2+marks3

    average=total/3

    if average >= 45:

        grade="Pass"

    else:

        grade="Fail"

    return total,average,grade

print(result(50,60,70))

#(180, 60.0, 'Pass')

Challenges on variable length arguments;

def unique\_number(\*args):

    num=set(args)

    return list(num)

num=input("Enter numbers")

# print(num.split())

# ['2', '6', '1', '2', '3', '5', '6', '7']

numbers=[int(n) for n in num.split()]

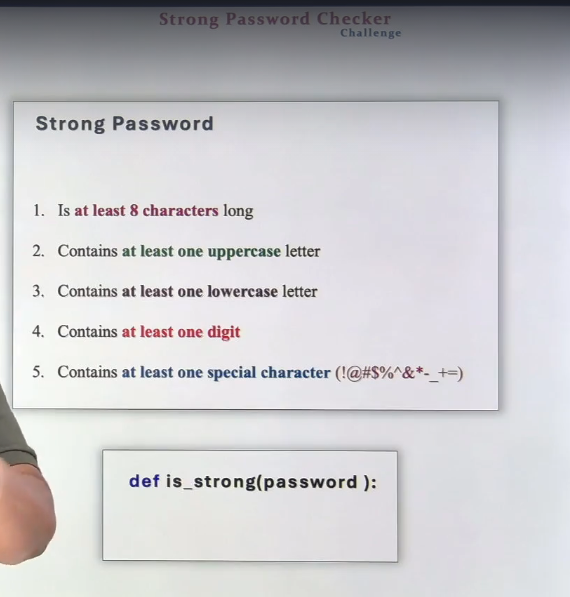
# print(numbers)

# [1, 2, 0, 3, 5, 4, 3]

unique=unique\_number(\*numbers)

print(unique)

Strong password checker:



def is\_strong(password):

    msg = 'Password must contain at least'

    if len(password) < 8:

        return False, msg + " 8 characters"

    has\_upper = any(c.isupper() for c in password)

    print(has\_upper)

    #True

    has\_lower = any(c.islower() for c in password)

    has\_digit = any(c.isdigit() for c in password)

    special\_chars = set("!@#$%^&\*-\_+=")

    print(special\_chars)

    #True

    has\_special = any(c in special\_chars for c in password)

    print(has\_special)

    #True

    if not has\_upper:

        return False, msg + " one uppercase letter."

    if not has\_lower:

        return False, msg + " one lowercase letter."

    if not has\_digit:

        return False, msg + " one digit."

    if not has\_special:

        return False, msg + " one special char (!@#$%^&\*-\_+=)"

    return True, "Password is strong!"

password = input("Enter your password: ")

valid, message = is\_strong(password)

print(message)