Variable length

pt()
pt(10)
pt(10, 12.5)
pt(10, 12.5, "hello")

→ pt function can take Variable length args

def fun(*args):

1. *args → is used for Variable length args
2. Tuple is created for Var length args

```
def fun(*args):
    pt(args)
    #pt(typ(args)) #class type
    for i in args:
        pt(i, end=" ")
```

fun(5) #5
fun(5, 10) #5, 10
fun(5, 10, 15) #5, 10, 15
fun(5, 12.5, "hello", 15)

```
def fun(*args):
    for x in args:
        if type(x) == int:
            print(x)
#10, 15
fun(10, 12.5, "we110")
```

3. fun(a, b, *args)
   a and b are positional args 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)
#Syntax Error
fun(*args, a, b)
a, b are keyword args only

def fun(*args, a, b)
    print(a, b, args)
fun(10, 20, 30, 40, 50)  # TypeErrors
fun(10, 20, 30, a=40, b=50)  keyword-only
                40, 50, (10, 20, 30)
```

5.

fun(*L) → unpacking list elements

```
def fun(*args):
    pt(args, len(args))
L1 = [10,20,30]
fun(L1)
  ([10,20,30],)1
```

```
L1 = [10,20,30]
fun(*L1)
(10,20,30) 3
```

Variable length x position x Arguments.

Variable length args 2

1. *args is used for variable length args
2. Tuple is created for variable length args
3. fun(a,b,*args)
   a, b → positional
4. fun(*args, a,b) → keyword (a, b)
5. fun(*L1) → unpacking actual arguments

Variable length arguments x keyword

```
def fun(**kwargs):
    pt(kwargs)
```

a. **kwargs is used for variable length keyword args

b. Dictionary is created for kwargs

Variable length args → keyword

```
def fun(**kwargs):
    pt(kwargs)
    pt(type(kwargs)
fun(a=5, b=10, c=15)
```

{'a': 5
 b: 10
 c: 15}
<dict>

Prog 2:-

```
def fun(**kwargs):
    for item in kwargs.items():
        if item[0] == 'b':
            pt(item[1])

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

10

c. fun(**kwargs, a, b)

→ Arguments not passed after keyword args

```
def fun(**kwargs, a, b):
    pt(kwargs)

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

|| syntax error

d. fun(a, b, **kwargs)

a, b +ve | keyword args

```
def fun(a, b, **kwargs):
    pt(kwargs)

fun(a, b, kwargs)
```

pt(a, b, kwargs)

fun(a=5, b=10) { }

fun(a=5, b=10, c=15) # {'c': 15}

fun(1, 2, c=3) # 1, 2, {'c': 3}

works both ~~fn~~ keyword / positi

Addl: (3) Variable length

e. fun (*args, **kwargs).
Always "*args" will be first

f. fun(**args, a, b, **kwargs)
a, b → keyword
no positional.

def fun(*args, a, b, **kwargs)
pt(a, b, kwargs)
pt(args)

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

fun(1, 2, a=3, b=4, x=5, y=10):
pt(args) → 1, 2
pt(a, b, kwargs) → 3, 4, {'x': 5, 'y': 10}

↳ **kwars :- Used for Variable length Keyword args
Dictionary :- Created
fun(**kwars, a, b)
fun(a, b, **kwargs)
a and b can be positional keyword

fun(*args, **kwargs)
fun(*args, a, b, **kwargs) → args should be left
→ a and b should keyword only.

X Args not allowed after **kwargs

Multiple Returns

```
def sum(a,b,c):
    sum = a+b+c
    prod = a*b*c
    return sum, prod
pt(sum(5,10,15))
# (30,750) #tuple
```

prog 2

```
def result(m1,m2,m3):
    total = m1+m2+m3
    avg = total/3
    if avg >= 45:
        grade = 'pass'
    else:
        grade = 'fail'
    return total, avg, grade
pt(result(50,60,70))
# (180,60.0,'pass')
```

Unique numbers

```
def unique(*args):
    num = set(args)
    return list(num)

num = input("  ")
# num.split()
nums = [int(n) for n in num.split()]
unique = unique-no(*numbers)
print(unique)
```

Strong password checker

```
def is_strong(password):
    msg = 'password must'
    if (len(password) < 8:
        return False, msg + '8 ch'
    has_upper = any(c.isupper() for c in passw
                    # True
    has_lower = any(c.islower() ...)
    has_digit = any(c.isdigit()
    spec_ch = set("  ")
    has_spec = any(c in spec_ch for c in
                   password)
```

Valid Args ✓

```
if not has_upper:
    return False, msg + "
if not has_lower:
    return False, msg + "
if not has_digit:
    return False, msg + "
if not has_spec:
    return False, msg + "
return True, "password is strong")

passwd = input("                r")
Valid, message = is_strong(password)
print(Message)
```