Python allows defining function with 3 types of parameters/arguments

1. Function with required arguments
    a. Function with required positional arguments
    b. Function with required keyword arguments
2. Function with default arguments
3. Function with variable length arguments/arbitrary arguments
    a. Function with variable length positional arguments
    b. Function with variable length keyword arguments

Function with required arguments

Function with required arguments required values at the time of calling or invoking function.
Based on the values sending to the function, these required arguments are two types.

1. Required positional arguments
2. Required keyword arguments

Syntax:

```
def function-name(parameter1,parameters2,….):
        statement-1
        statement-2
```

Example

```
def fun1(a,b):
   print(a,b)


fun1(10,20)  # Postional arguments
fun1(a=100,b=200) # keyword arguments
fun1(b=300,a=400)
```

Output

```
10 20
100 200
400 300
```

Required positional only arguments
If a function is defined with required positional arguments, function must called by sending values using position.

Syntax:
def <function-name>(param1,param2,param3,/):
    statement-1
    statement-2

Example:
```
def fun2(a,b,/):
   print(a,b)

fun2(100,200)
#fun2(a=100,b=200) Error
```

Output
100 200

Required keyword only arguments
If function is defined with keyword only arguments, function must be called by sending values with parameter names/argument names.

Syntax:
def <function-name>(*,param1,param2,…):
    statement-1
    statement-2

Example:
```
def fun3(*,a,b):
   print(a,b)

def fun4(x,y):
   print(x,y)
```

```
fun3(a=100,b=200)
#fun3(1000,2000)
fun4(10,20)
fun4(x=100,y=200)
```

Output

```
100 200
10 20
100 200
```

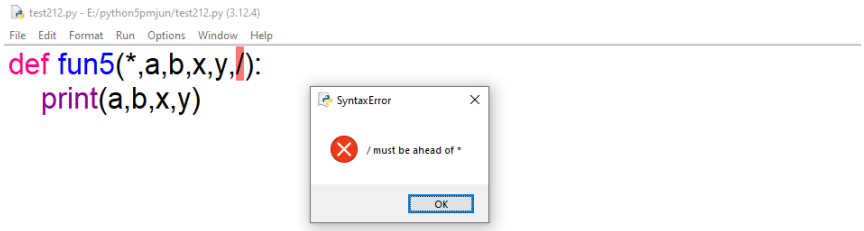
Example

```
def fun5(a,b,/,*,x,y):
    print(a,b,x,y)

fun5(10,20,x=30,y=40)
fun5(100,200,x=300,y=400)
```

output

```
10 20 30 40
100 200 300 400
```



The above code generate syntax error because
Positional arguments followed by keyword arguments.

Example:

```
def is_even(num):
    if num%2==0:
        print(f'Even Number')
    else:
```

```
        print(f'Odd Number')


is_even(5)
is_even(6)
res=max(10,20,30,40,50)
print(res)
```

Output

```
Odd Number
Even Number
50
```

return keyword

return is branching statement or keyword in python.
This statement is used inside function to return value.
Return statement after returning value, it terminates execution of function.
Return statement is used to return only one value/object.

```
def <function-name>(param,param,param,…):
        statement-1
        statement-2
```

Example:

```
def is_odd(num):
    if num%2!=0:
        return True
    else:
        return False

def is_prime(num):
    c=0
    for i in range(1,num+1):
        if num%i==0:
```

```
            c=c+1
    return c==2

a=is_odd(7)
print(a)
b=is_odd(8)
print(b)
c=is_prime(5)
print(c)
d=is_prime(8)
print(d)
```

Output

```
True
False
True
False
```

Example:

```
def fun1():
    print("Hello")
    return
    print("Bye")

def fun2():
    return 10,20,30,40,50

def fun3():
    return 10
    return 20
    return 30


fun1()
x=fun2()
print(x)
y=fun3()
```

```
print(y)
```

Output

```
Hello
(10, 20, 30, 40, 50)
10
```

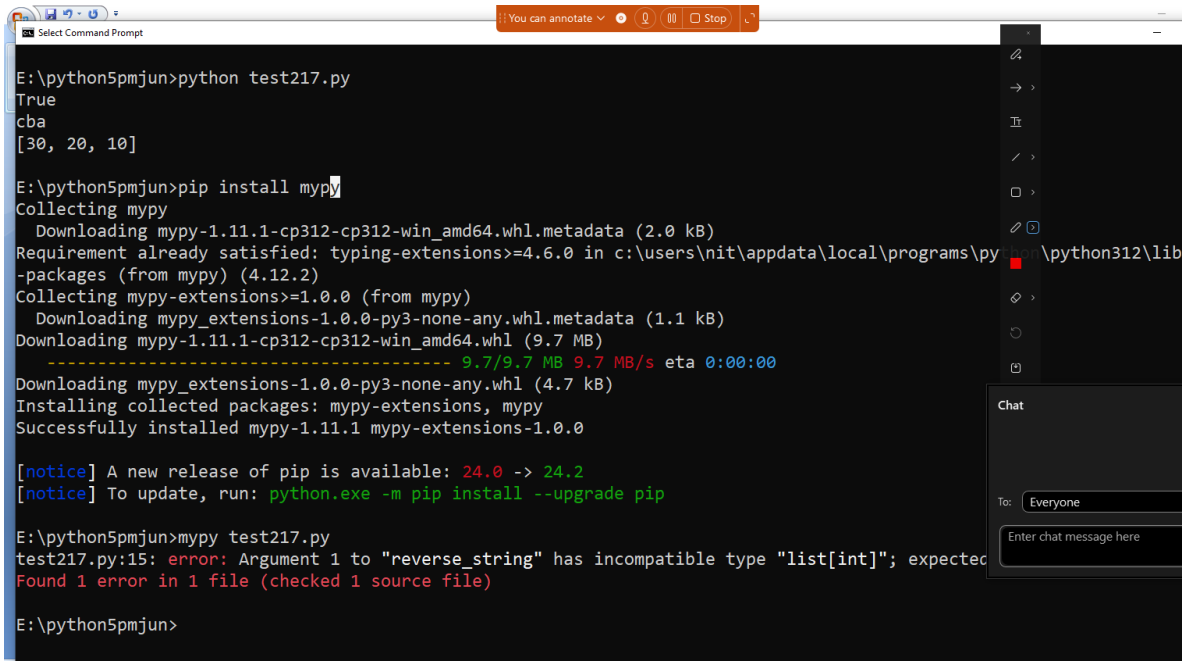
Type hints

Python is a dynamically typed language, which means you never have to explicitly indicate what kind of variable it is. But in some cases, dynamic typing can lead to some bugs that are very difficult to debug, and in those cases, Type Hints or Static Typing can be convenient. Type Hints have been introduced as a new feature in Python 3.5.

Example:

```
def is_odd(num:int)->bool:
    if num%2!=0:
        return True
    else:
        return False
def reverse_string(s:str)->str:
    s=s[::-1]
    return s


x=is_odd(5)
print(x)
y=reverse_string("abc")
print(y)
z=reverse_string([10,20,30])
print(z)
```


You can annotate
Stop
Select Command Prompt
E:\python5pmjun>python test217.py
True
cba
[30, 20, 10]
E:\python5pmjun>pip install mypy
Collecting mypy
Downloading mypy-1.11.1-cp312-cp312-win_amd64.whl.metadata (2.0 kB)
Requirement already satisfied: typing-extensions>=4.6.0 in c:\users\nit\appdata\local\programs\python\python312\lib
-packages (from mypy) (4.12.2)
Collecting mypy-extensions>=1.0.0 (from mypy)
Downloading mypy_extensions-1.0.0-py3-none-any.whl.metadata (1.1 kB)
Downloading mypy-1.11.1-cp312-cp312-win_amd64.whl (9.7 MB)
---------- 9.7/9.7 MB 9.7 MB/s eta 0:00:00
Downloading mypy_extensions-1.0.0-py3-none-any.whl (4.7 kB)
Installing collected packages: mypy-extensions, mypy
Successfully installed mypy-1.11.1 mypy-extensions-1.0.0
[notice] A new release of pip is available: 24.0 -> 24.2
[notice] To update, run: python.exe -m pip install --upgrade pip
E:\python5pmjun>mypy test217.py
test217.py:15: error: Argument 1 to "reverse_string" has incompatible type "list[int]"; expected
Found 1 error in 1 file (checked 1 source file)
E:\python5pmjun>
Chat
To:
Everyone
Enter chat message here