Flow of Execution in a Function Call

Flow of Execution in a Function Call

- **Programs to add two number through a function**

```
# program add.py to add two numbers through a function
def calcSum (x, y) :
    s = x + y                                   # statement 1
    return s                                    # statement 2
num1 = float(input( "Enter first number :" ) )      # 1 (statement 1)
num2 = float(input( "Enter second number :" ) )     # 2 (statement 2)
sum = calcSum(num1, num2)                           # 3 (statement 3)
print("Sum of two given numbers is", sum)           # 4 (statement 4)
```

016 PROGRAM TO ADD TWO NUMBERS

```
def calcsum(x,y):  1 usage
    sum= x+y
    return sum


num1=float(input("Enter first number: "))
num2=float(input("Enter second number: "))
sum=calcsum(num1,num2)
print("sum of two given numbers is",sum)
```

Flow of Execution in a Function Call

- Program execution begins with the first statement of _main_ sagments.
- Def are also read but ignored until called.
- it will become clear to you a few moments just read on

NOTE : The flow of education refers to the order in which statements are executed during a program run.

017 TWO NUMBERS FOR SUM

```
x=(int(input("enter the first number for the sum:")))
y=(int(input("enter the second number for the sum:")))
def sum(x,y):
    s=x+y
    return s
print(sum(x,y))
```

Flow of Execution in a Function Call

- **Arguments and parameters**
- As you know that you can pass values two functions.
- For this, you define variables to receive values in function definition and you send values via and function call statement.
- For example,

018 FUNCTION WITH ARGUMENTS

```
def multiply (a, b):  5 usages
    print(a * b)

y = 3
multiply( a: 12, y)
multiply(y, y)
x = 5
multiply(y, x)
```

Flow of Execution in a Function Call

- Arguments in Python can be one of these value types:
  - Literals
  - Variables
  - Expressions

Flow of Execution in a Function Call

```
multiply( 3, 4 )          # both literal arguments
p = 9
multiply( p, 5)           # one literal and
                          # one variable argument
multiply( p, p + 1)       # one variable and
                          # one expression argument
```

019 FUNCTION WITH ARGUMENTS

```
def multiply (a, b):  5 usages
    print(a * b)

p = 9
multiply(p, b:5)
multiply(p, p+1)
```