HO CHI MINH CITY UNIVERSITY OF TRANSPORT
FACULTY OF INFORMATION TECHNOLOGY
SOFTWARE ENGINEERING DEPARTMENT

CHAPTER 8
Functions And Program Organization

CONTENTS

Problem

Example 1:

- Suppose that you need to find the sum of integers
  - from 1 to 10, 20 to 37, and 35 to 49
- How do you write a program to solve this problem?

Example 2:

- Given two integers n and k. Find the binomial coefficient

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

- How do you write a program to solve this problem?

Problem: Repeat a particular task many times → very time consuming and very inefficient.

1. Organizing Program

- Our programs are going to start getting bigger and more complicated.

→ organize them in smaller pieces so they're easier to write, read and modify later.

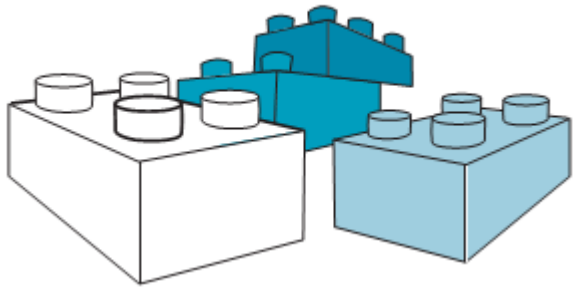
- There are three main ways to break programs into smaller parts:
  - **Functions - Procedural programming**
  - Modules - Modular programming
  - Objects – Object oriented programming

Functions - Procedural programming

- A **function** is a block of code that performs a specified task.
- A function is a subprogram to build a bigger program.
- This is the solution that give a **name to a piece of code.**

→*Every time that you need it, just call it.*

- **Benefits of breaking a program into functions**:
  - Code re-use within a program or across multiple programs
    → **Write one time, use many times**.
  - Smaller, simpler functions are easier to read, understand and maintain.
  - Reduce complexity of the program.
  - Hiding implementation details from users.



Example

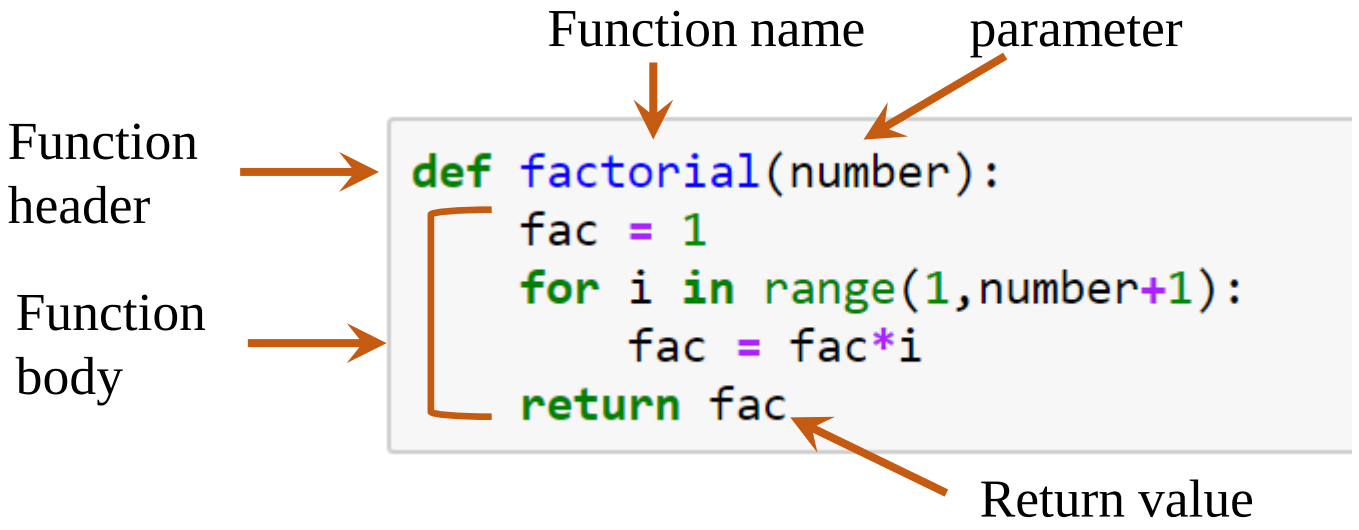
In [12]:
```python
# Define function to find factorial of a number
def factorial(number):
    fac = 1
    for i in range(1,number+1):
        fac = fac*i
    return fac

#Find the binomial coefficient
n = int(input("n = "))
k = int(input("k = "))
if n <= k:
    binom = 0
else:
    binom = factorial(n)//(factorial(k)*factorial(n-k))
print("Binomial coefficient:",binom)
```

```
n = 10
k = 5
Binomial coefficient: 252
```

2. Defining a Function

- The syntax for defining a function is as follows:

```
def functionName(list of parameters):
    <Function body>
```



Defining a Function

NOTE:

- Data can be passed to functions as parameter
- Parameters are optional, separated by commas. A function may not have any parameters.
- "**return**" statement:

return [<expression>]

  - Used to return a value.
  - <expression>: is optional.
  - The function terminates when a return statement is executed.
  - Some functions perform desired operations without returning a value

Examples

```
1  def my_sum(a,b,c):
2      return a + b + c
3
4  print(my_sum(1,2,3))
```

```
6
```

```
1  def my_abs(val):
2      if val < 0:
3          return 0 - val
4      return val
5
6  x = my_abs(-2.7)
7  print(x)
```

```
2.7
```

```
1  def printMyAddress():
2      print("123 Main Street")
3      print("Ottawa, Ontario, Canada")
4
5  printMyAddress()
```

```
123 Main Street
Ottawa, Ontario, Canada
```

3. Calling a Function

- Defining a new function does not make the function run.
  → To use a function → have to call it.
  → Calling a function executes the code in the function.
- Syntax of a function call:

```
functionName(list of arguments)
```

  - *Arguments:* are assigned to the parameters in the function definition
- There are two ways to call a function:
  1. If the *function returns a value* → a call to that function is usually treated as *a value*
  2. If a *function does not return a value* → the call to the function must be *a statement*.

Calling a Function

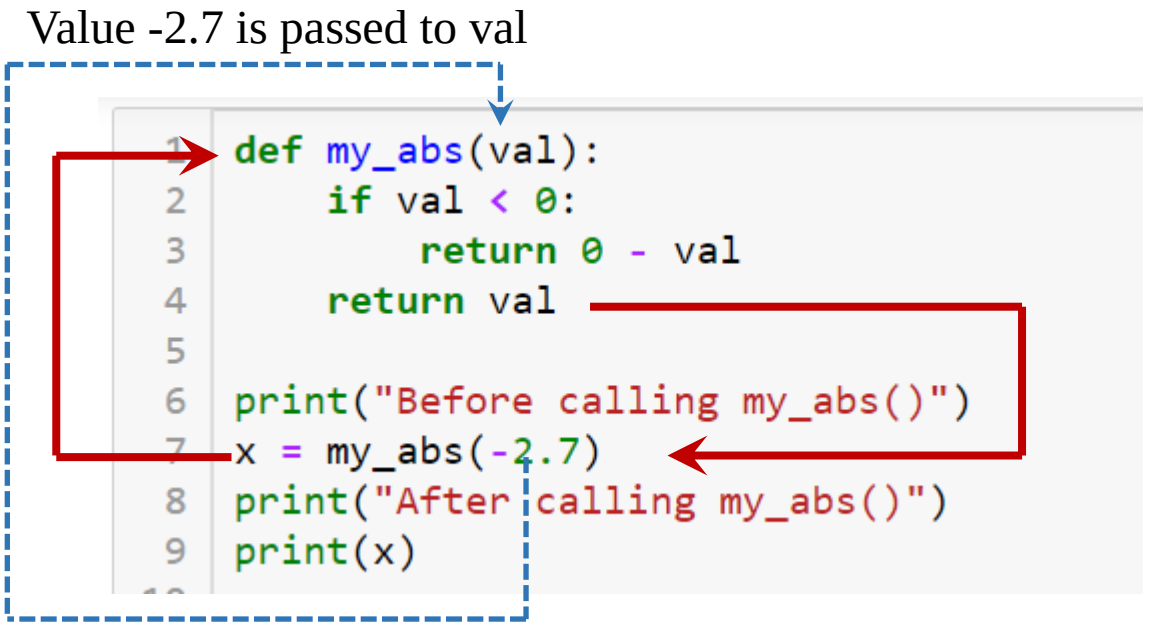
```
6  x = my_abs(-2.7)
7
8  y = my_abs(-2.7) + 1
9
10 print(my_abs(-2.7))
```

```
6  a,b,c = eval(input("Enter three numbers"))
7  if my_sum(a,b,c) > 100:
8      print("Sum is greater than 100!")
9  else:
10     print("Sum is not greater than 100!")
```

```
1  def printMyAddress():
2      print("123 Main Street")
3      print("Ottawa, Ontario, Canada")
4
5  printMyAddress()
```

Calling a Function

When a function is called, the program control jumps to that function definition and executes the statements inside the function body.



After executing the body of the function, the program control jumps back to where the function was called.

Functions without Return Values

- Some functions do not return any value.

  → By default, **None** is returned automatically

  → **return** statement can be omitted, but it can be used for terminating the function.

```python
# This function does not return any value
def isPass(score):
    if score < 0 or score > 10:
        print("Invalid score")
        return # terminate the function
    if score >= 5.0:
        print('Pass')
    else:
        print('Failed')
# end function

isPass(-1)
print(isPass(4.9))
```

```
Invalid score
Failed
None
```

Returning Multiple Values

- Syntax:

  return val1, val2, val3, ..., valN

- When calling a function returning multiple values, the number of variables on the left side of = operator must be equal to the number of values returned.

```
1  def my_divmod(a,b):
2      div = a//b
3      mod = a%b
4      return div,mod
5
6  d,m = my_divmod(5,2)
7  print(d,m)
```

```
2 1
```

4. Positional and Keyword Arguments

- There are two kinds of arguments:
  - Positional arguments: must match the parameters in order, number, and type.
  - Keyword arguments: passing each argument in the form

    key = value

- **Note**:

Positional arguments must appear before any keyword arguments.

```
1  def display(message, n, line):
2      for i in range(n):
3          print(message,end=" ")
4      size = len(message)*n
5      print("\n",line*size)
6
7  display("Python",3,"*")
8  display(n = 3,message = "Python",line = "-")
9  display("Python",n = 3,line = ".")
10 #display(n = 3,"Python",line = ".") --> Error!!!
```

Positional arguments

Keyword arguments

```
Python Python Python
 ******************
Python Python Python
 ------------------
Python Python Python
 ..................
```

5. Passing Arguments

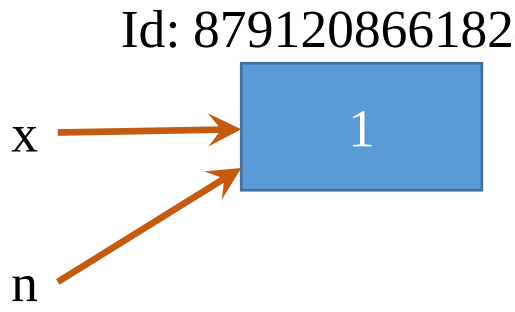
- All data are objects in Python, a variable for an object is actually a reference to the object.
- When you call a function with arguments:
  - Value of an argument is passed to a parameter → This mechanism is known as **Pass By Value.**
  - This value is actually a reference value to the object.
- **Note**:
  - If argument is immutable, the changes made to the parameter will not affect the argument.
  - If the argument is mutable, the changes made to the parameter variable will affect the argument.

Passing Arguments

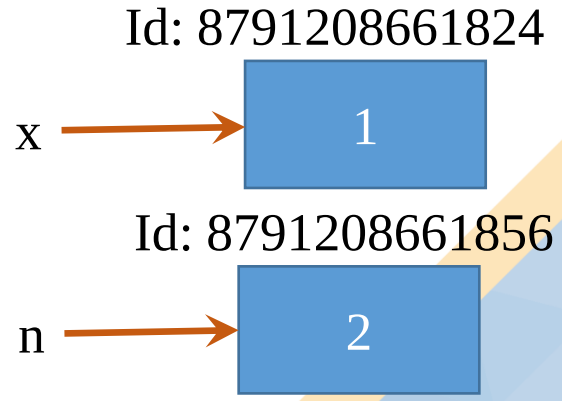
```
1  def increment(n):
2      n += 1
3      print("n =",n)
4
5  x = 1
6  print("Before calling increment(), x =",x)
7  increment(x)
8  print("After calling increment(), x =",x)
```

```
Before calling increment(), x = 1
n = 2
After calling increment(), x = 1
```

Call increment()



n += 1



Passing Arguments

```python
def add(ds):
    ds.append(4)
    print("Inside function: ",ds)

# [1,2,3] is a mutable object
my_list = [1,2,3]
print("Before function call: ",my_list)
add(my_list)
print("After function call:",my_list)
```

```
Before function call:  [1, 2, 3]
Inside function:  [1, 2, 3, 4]
After function call: [1, 2, 3, 4]
```

6. Scope of Variables

- The **scope of a variable** is the part of the program where the variable can be referenced.
- **Local variable**: created inside a function
  - Can only be accessed within a function.
  - Scope: starts from its creation and continues to the end of the function that contains the variable.
- **Global variable**: created outside all functions
  - Accessible to all functions
  - Scope: starts from the point they are defined and continues to the end of the program

Scope of Variables

Example 1:

```
1  x = 1 # x: Global variable
2  def func1():
3      y = x + 1 # y: Local variable
4      print(y)
5
6  func1()
7  print(x)
8  # print(y) -- Out of scope --> Error
```

```
2
1
```

Example 2:

```
1  x = 1 # Global variable
2  def func2():
3      # x = x + 1 --> Error
4      x = 2 # New local variable
5      print(x)
6
7  func2()
8  print(x)
```

```
2
1
```

Example 3:

```
1  x = eval(input("Enter a number: "))
2  if x > 0:
3      y = 1
4  print(y) # --> Error if x <= 0
```

```
Enter a number: 2
1
```

Example 4:

```
1  sum = 0
2  for i in range(5):
3      sum += i # i is created in the loop
4  print(i) # i accessed outside the loop
```

```
4
```

7. Default Parameter

- Specify a ***default value*** *for one or more parameters*.
- This creates a function that can be called with fewer arguments than it is defined to allow.
- **Note**:
  - The non-default parameters must be defined before default parameters.

Default Parameter

```python
def calArea(width = 1, height = 1):
    area = width*height
    print(f"width = {width}, height = {height}, area = {area}")

calArea()
calArea(2,3)
calArea(height=4, width=5)
calArea(5)
calArea(height=4)
```

```
width = 1, height = 1, area = 1
width = 2, height = 3, area = 6
width = 5, height = 4, area = 20
width = 5, height = 1, area = 5
width = 1, height = 4, area = 4
```