06.함수

함수

모든 프로그래밍의 문법은 개발자가 개발 생산성을 향상할수 있고 효율적이고 안정성 있는 개발을 할 수 있도록 고안되고 발전된다.

- 함수 (재사용성과 유지보수성의 증가)

. 반복적인 코드 를 작성해 두고 언제든지 재 사용 할 수 있도록 할 수 있다.

. 같은 로직을 반복하나 함수를 실행할때 투입하는 값에 따라 다른 결과를 얻을 수 있다.

. 함수를 수정시 사용하는 모든 로직에서 일괄 적용 된다. 모든 내용을 수정 할 필요 없이 함수만 수정함으로서 수정 이 용이하다 (유지보수성)

- 일반적인 로직의 작성

. 순차적인 정수 를 누적합산하는 값을 구하는 문제가 있다고 할때
아래와 같이 누적 합산 로직을 하나하나 작성하여 프로그램을 구현 할 수 있다.

Chap01_intro > 06.Function[함수].py > ...

```
1  sum = 0
2  for num in range(5):
3      sum += num
4  print('~ 4 = ',sum) # 0 , 1 , 2 , 3 , 4 의 모든 합을 나타낸다.
5
6  sum = 0
7  for num in range(6):
8      sum += num
9  print('~ 6 = ',sum) # 0 , 1 , 2 , 3 , 4 , 5  의 모든 합을 나타낸다.
10
11 sum = 0
12 for num in range(7):
13     sum += num
14 print('~ 7 = ',sum) # 0 , 1 , 2 , 3 , 4 , 5 , 6 의 모든 합을 나타낸다.
15
16
17 sum = 0
18 for num in range(8):
19     sum += num
20 print('~ 8 = ',sum) # 0 , 1 , 2 , 3 , 4 , 5 , 7 의 모든 합을 나타낸다.
```

> sum 변수 초기화
> 반복 문 조건
> 반복 실행
> 결과 출력

> sum 변수 초기화
> 반복 문 조건
> 반복 실행
> 결과 출력

> sum 변수 초기화
> 반복 문 조건
> 반복 실행
> 결과 출력
.
.
.

> 결과

```
Microsoft Windows [Version 10.0.22621.2134]
(c) Microsoft Corporation. All rights reserved.

D:\Python>C:/Users/MasterD/AppData/Local/Programs/Python/Python311/
~ 4 =  10
~ 6 =  15
~ 7 =  21
~ 8 =  28
```

. 만약 각 누적 합산의 결과 를 2 부터 100 까지의 누적 합산 결과를 모두 보고 싶을때
위 반복되는 로직을 99 건 작성 하여야 한다.

```
sum = 1
for num in range(3):
    sum += num
print('~ 2 = ',sum) # 0 , 1 , 2   의 모든 합을 나타낸다.
#.....................................
sum = 1
for num in range(101):
    sum += num
print('~ 100 = ',sum) # 0  ~  100 의 모든 합을 나타낸다.
```

. 만약에 로직을 잘못 작성하여 누적 곱으로 표현 해야 할 경우.
> 반복되는 모든 로직을 모두 수정 하여야 한다.

```python
sum = 1
for num in range(1,5):
    sum *= num
print('~ 4 = ',sum) # 0 , 1 , 2 , 3 , 4 의 모든 곱을 나타낸다.

sum = 1
for num in range(1,6):
    sum *= num
print('~ 6 = ',sum) # 0 , 1 , 2 , 3 , 4 , 5  의 모든 곱을 나타낸다.

sum = 0
for num in range(1,7):
    sum += num
print('~ 7 = ',sum) # 0 , 1 , 2 , 3 , 4 , 5 , 6 의 모든 곱을 나타낸다.


sum = 1
for num in range(1,8):
    sum *= num
print('~ 8 = ',sum) # 0 , 1 , 2 , 3 , 4 , 5 , 7 의 모든 곱을 나타낸다.
```

> 결과 ?

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Microsoft Windows [Version 10.0.22621.2134]
(c) Microsoft Corporation. All rights reserved.

D:\Python>C:/Users/MasterD/AppData/Local/Programs/Pyt
~ 4 =  24
~ 6 =  120
~ 7 =  21
~ 8 =  5040
```

< 모든 항목을 수정을 완료 하였으나 실수로 세번 째 로직을 수정 하지 못하여 의도와는 다른 결과를 도출한다. >

- 함수 의 생성 과 호출

. 위의 예제 처럼 비효율적이고 생산적이지 못한 상황을 해결하려고 등장한 문법
. 반복적인 코드 를 묶어 두고 언제든지 재 사용 할 수 있도록 할 수 있다.
. 함수 의 선언

```python
def 함수이름(인자) :
```

. 함수 의 호출

```python
함수이름('인수')
```

* 인수 : 함수 를 호출 할때 함수에게 전달 하는 값
* 인자 : 함수 를 생성 할때 전달 받을 변수

. 함수 의 특징
> 인수 및 인자 가 없는 경우
* 함수를 호출하면 '안녕하세요' 를 출력

```
Chap01_intro > 06.Function[함수].py > ...
1  def MyFunction() :
2      print('안녕하세요')
3
4  MyFunction()

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Microsoft Windows [Version 10.0.22621.2134]
(c) Microsoft Corporation. All rights reserved.

D:\Python>C:/Users/MasterD/AppData/Local/Programs/
안녕하세요
```

> 인수 와 인자 가 있는 경우
 * 인수 의 값에 따라 다른 결과 를 출력한다.

```
Chap01_intro > 06.Function[함수].py > MyFunction > parameter
1  def MyFunction(parameter) :
2      print(parameter)
3
4  MyFunction(500)

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Microsoft Windows [Version 10.0.22621.2134]
(c) Microsoft Corporation. All rights reserved.

D:\Python>C:/Users/MasterD/AppData/Local/Programs/Python/Python
500
```

> 1행 : parameter 변수 를 인자 로 받는 MyFunction 함수
> 2행 : paramrter 변수의 값을 출력

> 4행 : MyFunction 함수 에 500 값을 인수로 전달하며 호출

> 결과 : 500 출력

> 잘못된 함수의 호출 1
 * 함수를 호출 하기 위해서는 함수 생성 로직이 선행 되어야 한다. (한줄한줄 차근차근 실행하는 인터프리터 언어이기 때문)

```
Chap01_intro > 06.Function[함수].py > MyFunction
1  MyFunction(500)
2
3  def MyFunction(parameter) :
4      print(parameter)
```

< 함수 생성 과 호출 을 잘못 사용한 예 >

> 잘못된 함수의 호출 2
 * 약속 되지 않은 인수 와 인자를 사용 시 오류 가 발생한다.

```
Chap01_intro > 06.Function[함수].py > ...
1  def MyFunction(parameter) :
2      print(parameter)
3
4  MyFunction()

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Microsoft Windows [Version 10.0.22621.2134]
(c) Microsoft Corporation. All rights reserved.

D:\Python>C:/Users/MasterD/AppData/Local/Programs/Python/Python311/python.exe
Traceback (most recent call last):
  File "d:\Python\Chap01_intro\06.Function[함수].py", line 4, in <module>
    MyFunction()
TypeError: MyFunction() missing 1 required positional argument: 'parameter'
```

< 인자를 받기로 선언한 함수를 인수 없이 호출하는 경우 오류 >

```
Chap01_intro > 06.Function[함수].py > ...
1  def MyFunction() :
2      print()
3
4  MyFunction('안녕하세요')

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Microsoft Windows [Version 10.0.22621.2134]
(c) Microsoft Corporation. All rights reserved.

D:\Python>C:/Users/MasterD/AppData/Local/Programs/Python/Python311/python.
Traceback (most recent call last):
  File "d:\Python\Chap01_intro\06.Function[함수].py", line 4, in <module>
    MyFunction('안녕하세요')
TypeError: MyFunction() takes 0 positional arguments but 1 was given
```

< 인자 를 받지 않기로 선언한 함수에 인수 를 전달할 경우 오류>

> 함수 의 인자 는 N 개 설정 할수 있다.

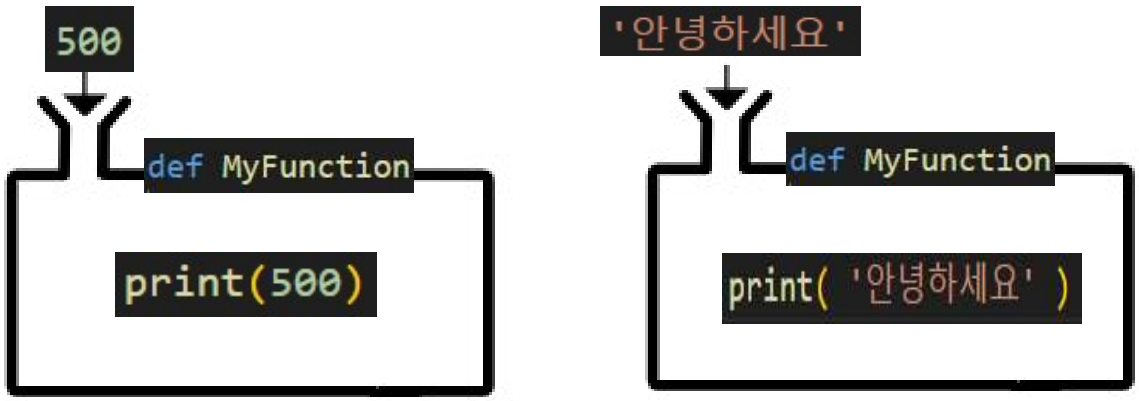
```
1  def Myfunction(value1, value2, value3) :
2      pass
```

< 인자 3 개 를 받는 함수 생성 >

- 투입 되는 값으로 로직을 수행 하고 종료되는 함수

. 함수를 실행하였을 경우 함수를 호출한 곳으로 반환 결과를 보내 주지 않는 함수

. MyFunction 을 호출 할때 전달 해 주는 값 에 따라 다른 결과를 표현하는 로직만 수행



- 함수 의 재사용성

. 첫번째 예 를 Addition 함수 를 만들어 반복 되는 구문을 작성하고 함수를 호출 하기

```
206  def Addition(Target) :
207      sum = 0
208      for Number in range(Target + 1):
209          sum += Number
210      print('~ ',Target,' = ',sum)
211
212  Addition(5)
213  Addition(6)
214  Addition(7)
215  Addition(8)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

D:\MyDoc\11.EDUCATION\1. Master Source\15.Python\0.MyFirstPython\0.Sou
scode\extensions\ms-python.python-2023.16.0\pythonFiles\lib\python\deb
hon\0.Source\Python\06.Function[함수].py" "
~  5  =  15
~  6  =  21
~  7  =  28
~  8  =  36
```

> 1행 : Target 변수를 인자로 가지는 Addition 함수 생성
> 2행 : Addition 함수의 지역변수 sum 에 0 초기화
> 3행 : 인자로 받은 Target 의 범위 만큼 반복 (0 ~ Target -1)
> 4행 : sum 변수에 0 부터 Target -1 까지 의 값을 누적 합산.
> 5행 : 결과 출력

> 7행 : Addition 함수 에 인수 로 5 값을 전달 후 수행
> 8행 : Addition 함수 에 인수 로 6 값을 전달 후 수행
> 9행 : Addition 함수 에 인수 로 7 값을 전달 후 수행
> 10행 : Addition 함수 에 인수 로 8 값을 전달 후 수행

* 반복적으로 수행 하는 로직을 하나의 집합으로 만들어 두고 필요할 경우 호출 하는 방식으로써 재사용성을 높이고 코드를 간결하게 구현 할 수 있다.

실습

- 위 함수 적용 로직을 응용하여 2 부터 100 까지의 누적 합산을 표현하는 로직을 함수로 간결하게 표현해 보세요

```
~  2  =  3
~  3  =  6
~  4  =  10
~  5  =  15
~  6  =  21
~  7  =  28
~  8  =  36
~  9  =  45
~  10  =  55
```

```
~  94  =  4465
~  95  =  4560
~  96  =  4656
~  97  =  4753
~  98  =  4851
~  99  =  4950
~  100  =  5050
```

- **함수의 유지 보수성**

. 함수의 로직 을 수정 할 경우

> 함수를 호출 하는 모든 로직이 일괄 수정된 내용이 적용 된다. (유지 보수성 증가)

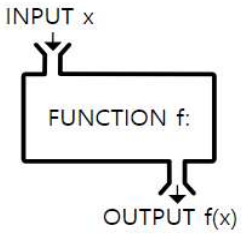
```
238  def Addition(Target) :
239      sum = 0
240      for Number in range(Target):
241          # 모든 로직에 인자값에 50을 추가로 더하는 구문이 삽입되는 경우
242          sum += Number + 50
243      print('~ ',Target,' = ',sum)
244
245  # 적용 된 내역을 함수를 호출하는 모든 로직에 일괄적으로 적용할수 있다.
246  Addition(5)
247  Addition(100)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

D:\MyDoc\11.EDUCATION\1. Master Source\15.Python\0.MyFirstPython\0.Source\Python> cmd /C "C:\Users\MasterD\AppData\
scode\extensions\ms-python.python-2023.16.0\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher 55602 --
hon\0.Source\Python\06.Function[함수].py" "
~  5  =  260
~  100  =  9950
```

* 생성한 함수에 범위의 모든 값에 50을 더해서 누적하는 구문으로 수정하여 함수를 호출하는 모든 구문이 일괄 적용 된것을 볼수 있다.

- **함수 의 결과 를 반환 (return)**

. 함수 를 실행 하면 함수의 결과 를 호출 한 곳으로 반환한다.



. 함수의 호출 과 반환의 예시

> calsum 이라는 함수 를 호출 시 인수를 전달 하고 결과를 반환 받음

```
Chap01_intro > 06.Function[함수].py > ...
1  def calcsum(end) :
2      sum = 0
3      for num in range(end +1):
4          sum += num
5      return sum
6
7  result = calcsum(10)
8  print(result)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Microsoft Windows [Version 10.0.22621.2134]
(c) Microsoft Corporation. All rights reserved.

D:\Python>C:/Users/MasterD/AppData/Local/Programs/Python/Py
55
```

> 7행 : calsum 함수 에 인수 10 값을 전달 하며 호출
>
> 1행 ~ 5행 수행
>
> 5행에서 반환한 결과 를 **result** 변수에 할당

> 8행 : 함수가 반환한 결과 를 출력

> 1행 : end 변수를 인자로 받는 calsum 함수 생성
> 2행 : 총 합을 누적 할 sum 변수 초기화
> 3행 : 인자로 받은 end 범위 까지 반복
> 4행 : sum 변수에 반복되는 수 누적 합산
> 5행 : 누적 된 결과 sum 의 값을 호출한 곳으로 **반환**

- 함수 와 인수/인자 의 응용 예제

```python
def avrage(value1, value2, value3) :
    return (value1 + value2 + value3) / 3

value1 = int(input('첫번째 정수값 : '))
value2 = int(input('두번째 정수값 : '))
value3 = int(input('세번째 정수값 : '))
print('3 값의 평균 은 ' ,avrage(value1,value2,value3) , ' 입니다')
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

D:\MyDoc\11.EDUCATION\1. Master Source\15.Python\0.MyFirstPython\0.Source\Python> cmd /C "C:\Users\MasterD\AppData\Local\Programs\Python\
scode\extensions\ms-python.python-2023.16.0\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher 65293 -- "D:\MyDoc\11.EDUCATION
hon\0.Source\Python\06.Function[함수].py" "
첫번째 정수값 : 10
두번째 정수값 : 20
세번째 정수값 : 30
3 값의 평균 은  20.0  입니다
```

- TIP

. 로직을 완성하지 않았지만 미완성 로직으로 인해 오류를 내고 싶지 않을때 Pass

> 로직의 작성을 지금 당장 하지 않고 추후 작성 을 할때는 pass 를 사용한다.

```python
for i in range(10) :
    pass

def MyFunction():
    pass
```

> pass 가 없이 함수 작성을 완료하지 않으면 오류 가 발생한다.

```python
for i in range(10) :

def MyFunction():
    pass
```

실습

1. 정수로 시작 과 끝의 범위 와 몇배수 인지 의 정보를 인자로 받는 함수 를 작성하여
   시작 과 끝 범위 내에 있는 배수 를 모두 표현하는 로직을 구현해 보세요.

```
시작 정수값 : 10
종료 정수값 : 200
찾을 배수 : 3
3  배수의 결과 :  12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 63, 66, 69, 72, 75, 78, 81, 84, 87, 90, 93, 96, 99, 102, 105, 108, 111,
 114, 117, 120, 123, 126, 129, 132, 135, 138, 141, 144, 147, 150, 153, 156, 159, 162, 165, 168, 171, 174, 177, 180, 183, 186, 189, 192, 195, 198,
```

재귀 호출 함수 (recursion)

- 자신을 홀출 하도록 구현된 함수

. 함수가 종료되는 조건을 반드시 포함해야함.

. 프로그램이 간결해지고 가독성이 높아짐.

. 시간복잡도가(프로그램이 구현완료되는 시간) 증가하고 반복문을 사용할 때 보다 성능이 떨어질 수 있음.

- Factorial 을 재귀 호출 함수 로 풀어보기 (이진 트리 구조)
  . Factorial 이란 특정한 수의 모든 수열을 곱하는 수학 용어이다.
  . 수보다 작거나 같은 모든 양의 정수의 곱이다. n이 하나의 자연수일 때, 1에서 n까지의 모든 자연수의 곱을 n에 상대하여 이르는 말이다.
  ex) 5 Factorial (5!) = 1 * 2 * 3 * 4 * 5
  n! = n * n(-1)! 5! = 5 * 4! * 5! 은 총 5번 의 루프를 돌면서
  4! = 4 * 3! n * (n-1)! 을 반복 한다.
  3! = 3 * 2!
  2! = 2 * 1!
  1! = 1 * 0!

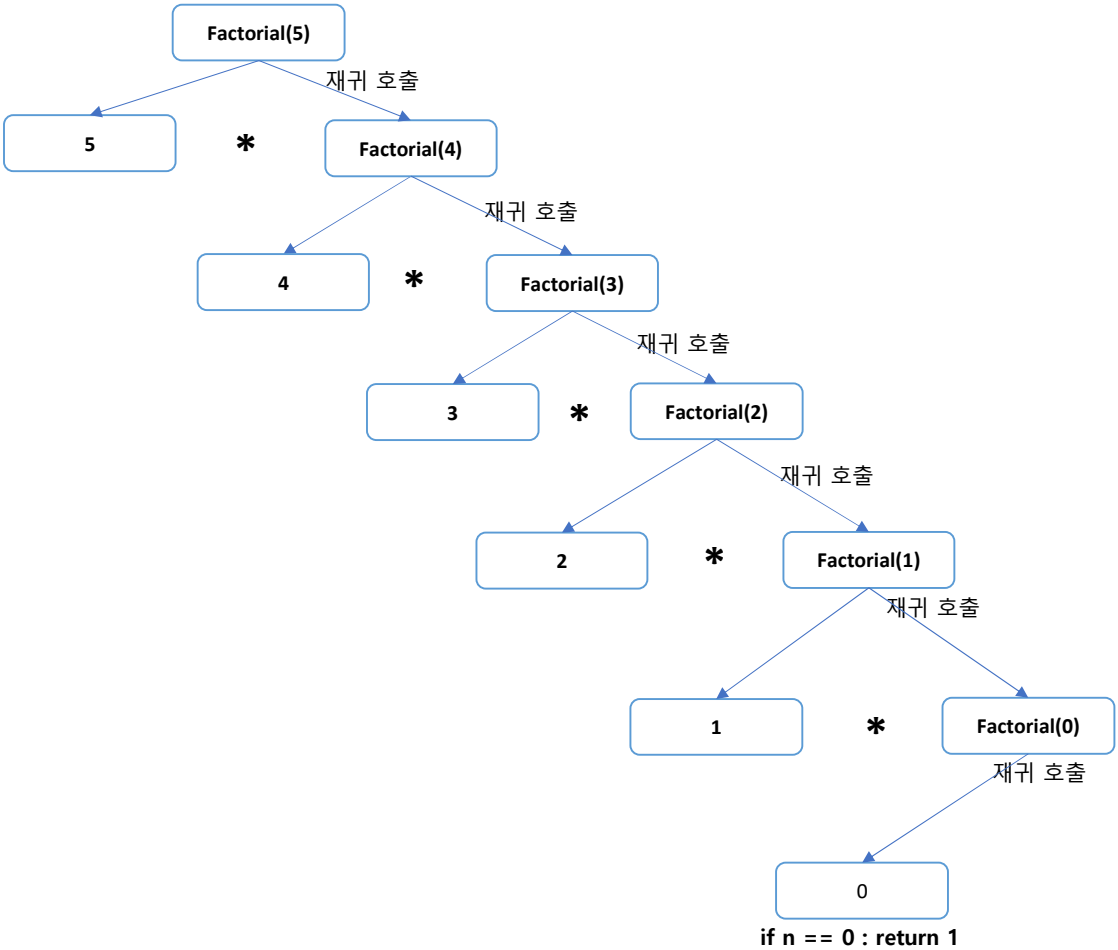
  . Factorial 프로그래밍 예제

```python
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

result = factorial(5)
print(result)  # 출력: 120
```

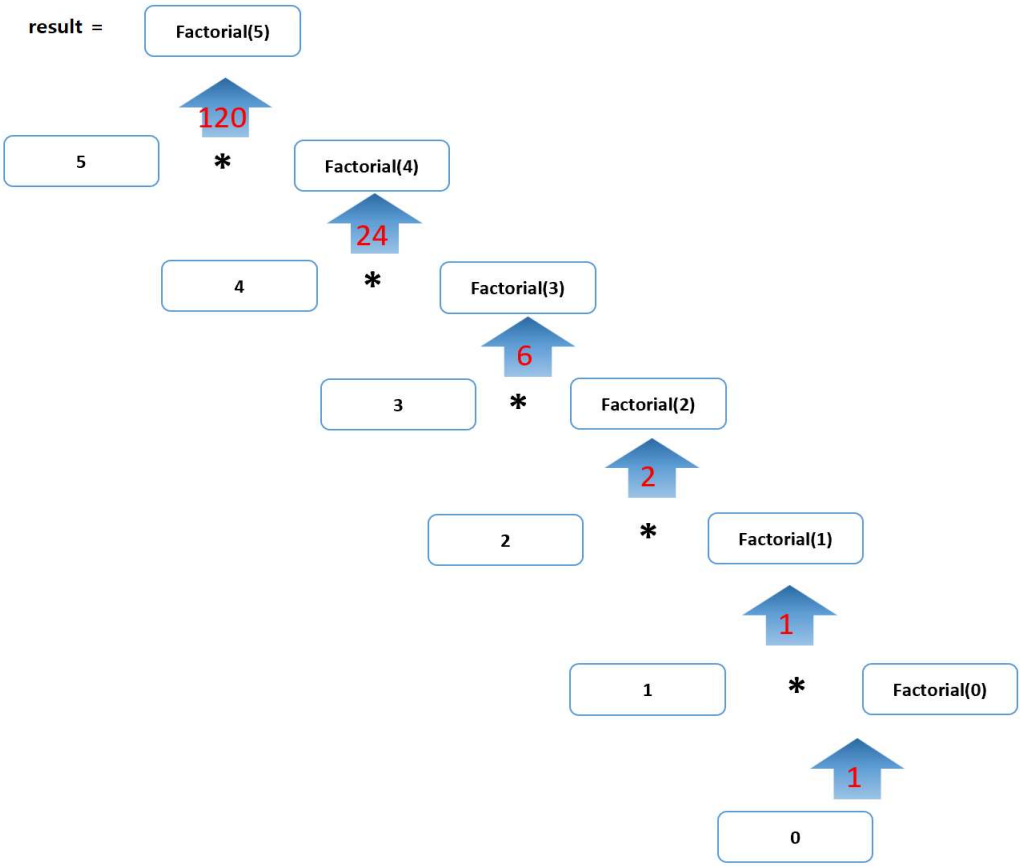
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

D:\Python>C:/Users/MasterD/AppData/Local/Programs/Python/Python311/pytho
120
```

>3행 : n 이 0 인 경우 1 을 반환한다. (재귀호출 의 종료 조건)
  * 재귀 호출 의 종료 조건이 없을경우 무한 재귀 루프에 빠지게 된다.
>5행 : n 의 수를 -1 하여 Factorial 을 구하는 자기자신의 함수를 호출한다.

> 프로그램의 실행



> 결과의 수렴


result =
Factorial(5)
120
5
*
Factorial(4)
24
4
*
Factorial(3)
6
3
*
Factorial(2)
2
2
*
Factorial(1)
1
1
*
Factorial(0)
1
0