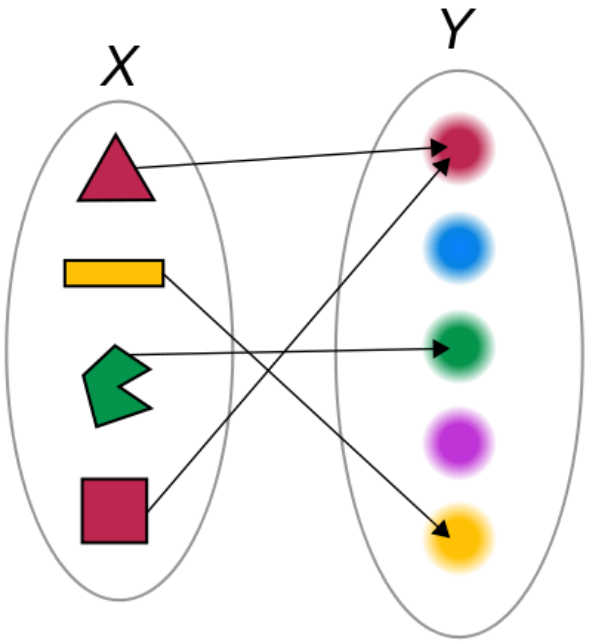
Fastcampus Web Programming SCHOOL

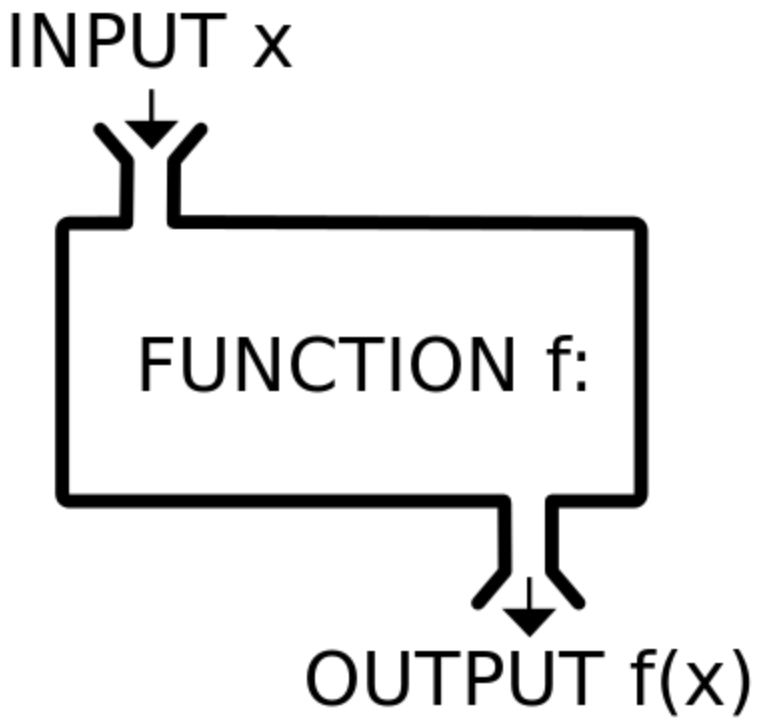
Python

function



- 수학적 정의: 첫 번째 집합의 임의의 한 원소를 두 번째 집합의 오직 한 원소에 대응시키는 대응 관계
- x: 정의역 y: 공역

function



- 프로그래밍에서의 함수: 입력값을 내부에서 어떤 처리를 통해 결과값을 출력하는 것

function

```
def function(parameter):
        실행문1
        실행문2
        ...
        return output
```

function

```
def awe_sum(a,b):
        result = a + b
    return result

a = 2
b = 3
print(awe_sum(a,b))
```

function without input

```
def print_hello():
    return "hello"

result_hello = print_hello()
print(result_hello)
```

function without return

```
def func_wo_return(a):
    print("This is function without return for " + str(a) + " times.")

func_wo_return()
```

function with multiple return

```
def mul_return(a):
    b = a + 1
        return a,b
```

return skill

```
def id_check(id):
        if id == "admin":
                print("invalid id: admin")
                return
        print("valid id: ", id)
```

parameter with initialize

```
def say_hello(name="Fool", nick=True):
        print("Hi, ", name)
        if nick == True:
                print("But, you are Fool")
        else:
                print("Oh, you are not Fool")
```

초기값을 설정할땐 항상 그 인자를 마지막에 두어야 합니다.

arguments

```
def mul_sum(*args):
        sum = 0
        for i in args:
                sum += i
        return sum
```

keyword arguments

```
def show_kwargs(**kwargs):
        print(str(kwargs))

show_kwargs(a=10, b="google")
```

keyword arguments

```
def kwargs_url(server, port, **query):
        url = "https://" + server + ":" + port + "?"
        for key in query.keys():
                url += key + "=" + query[key] + "&"
        return url

kwargs_url("localhost","8080", utm_source="google", keyword="naver")
```

variable outside function

```
a = "hello"
def glob_test(a):
        a += "world"
        return a

glob_test(a)
print(a)
```

```
a = "hello"
def glob_test(x):
        x += "world"
        return x

glob_test(a)
print(a)
```

variable outside function

```
def glob_test2(x):
    a += "world"
    x += "success"
        return x

glob_test2(a)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in glob_test2
UnboundLocalError:
local variable 'a' referenced before assignment
```

So, how to globalize

(1) using return

```
a = "hello"
def glob_test(a):
        a += "world"
    return a

a = glob_test(a)
print(a)
```

So, how to globalize

(2) use global

```
a = "hello"
def glob_test(a):
        global a
        a += "world"
    return a

glob_test(a)
print(a)
```

global 이라는 명령을 사용하여 전역변수로 사용하게 되면 함수는 독립성을 잃게 되어 함수가 외부변수에 의존적이게 됩니다.

Do it yourself!

Leap year

사용자로 부터 연도(0~9999 사이의 정수)를 입력 받아
4로 나뉘어 떨어지면 윤년,
100으로 나뉘어 떨어지면 평년,
400으로 나뉘어 떨어질땐 윤년
을 출력하는 함수를 작성하시오

Leap year(answer)

```
leap = False
def is_leap(y):
        if y % 4 == 0 and (y % 100 != 0 or y % 400 == 0)
                leap = True
        return leap

y = int(input("Is leap?? "))
print(is_leap(y))
```

numguess with function

```
def guesser(guess):
        if guess == answer:
                print("Correct! The answer was ", str(answer))
                break
        else:
                print("That's not what I wanted!! Try again!!")
```

Recursive

What is GNU?

- GNU is Not Unix
  - What is GNU?
  - GNU is Not Unix
    - What is GNU?
    - GNU is Not Unix
      - What is GNU?
      - GNU is Not Unix
        - ...

Recursive

```
times = int(input("How many times want to curse the beast??: "))
def recurse_beast(a):
        if a == 0:
                print("curse complete!")
        else:
                print("Fusion!!!(%d times left)" % a - 1)
                recurse_beast(a-1)

recurse_beast(times)
```

Fibonacci Sequence

Fibonacci Sequence

$$F_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F_{n-1} + F_{n-2} & n > 1 \end{cases}$$

Fibonacci Sequence with Recursion

```
def fib_rec(n):
    if n < 2:
        return n
    else:
        return fib_rec(n-1) + fib_rec(n-2)
```

Binet's Fibonacci formula

$$F_n = \frac{(1+\sqrt{5})^n - (1-\sqrt{5})^n}{2^n\sqrt{5}}$$

Binet's Fibonacci formula

```
import math


def fib_binet(n):
    sqrt_5 = math.sqrt(5)
    result = int(((1+sqrt_5)**n-(1-sqrt_5)**n) / (2**n*sqrt_5))
    return result
```

실행시간을 비교해봅시다.

- 40개의 피보나치 수 구하기
  - case 1:

    ```
    execution time: 90.98334097862244
    ```

  - case 2:

    ```
    execution time: 0.00013065338134765625
    ```

Fibonacci Recursion Flow

F_6 ->

F_5+F_4 ->

F_4+F_3 + F_3+F_2 ->

F_3+F_2 + F_2+F_1 + F_2+F_1 + F_1+F_0 ->

F_2+F_1 + F_1+F_0 + F_1+F_0 + 1 + F_1+F_0 + 1 + 1 + 0 ->

F_1+F_0+ 1 + 1+0 + 1+0 + 1 + 1+0 + 1 + 1 + 0 ->

1+0+ 1 + 1+0 + 1+0 + 1 + 1+0 + 1 + 1 + 0

= 8

Recursion 장점

- 재귀적 알고리즘 표현이 명확할 경우 Loop 사용보다 직관적인 코드
- 변수의 수를 줄이고, 가능한 경우의 수를 줄여줘 오동작 가능성이 줄어듦

Recursion 사용시 주의사항

- Python은 function depth가 1000으로 제한되며, 근접시 동작하지 않습니다.
- 시간복잡도를 감안해 Recursion을 작성해야 합니다.
- Recursion을 Escape할 장치를 마련해야 합니다.

Homework

사용자의 입력 `num` (0~950 사이의 정수)을 받아
1에서 `num` 까지의 모든 자연수의 곱(팩토리얼)을
Recursive, Iterative 두가지로 해결하세요.