[No. 4] **Flame of Passion** ____년 __월 __일

문제

인자로 주어지는 정수 이하의 모든 소수를 출력하는 명령 `\primes`를 작성하여라.

입력: `\primes{10}`
출력: 2, 3, 5, 7

다음과 같은 방법으로 소수를 구하자.

```
>>> def fnprimes(n):
        P=[]
        for i in range (2,n):
                isPrime = True
                for j in range (2, int(i ** 0.5) + 1):
                        if i % j == 0:
                                isPrime = False
                                break
                if isPrime:
                        P.append(i)
        return P

>>> fnprimes(20)
[2, 3, 5, 7, 11, 13, 17, 19]
```

1 fp 연산

TEX과 LATEX에 있는 "수"는 count와 dimension뿐이다. expl3는 fp 자료형을 도입하여 TEX에 현저히 부족한 실수 계산을 보완하려 하였다.

붙임 1: expl3가 처음 작성되기 시작하던 무렵에는 pgf의 계산 엔진을 활용하였다. 현재는 그렇지 않으나 이런 까닭에 pgfmath와 유사한 데가 남아 있다.

붙임 2: pgfmath 등장 이전에는 실수 계산을 위한 패키지들이 존재하였다. `realcalc`, `fp` 등이 이에 해당한다.

붙임 3: expl3의 fp는 유효숫자(significand) 16자리, 지수 ± 10000 범위의 수와, 부호 있는 0, 부호 있는 ∞를 포함한다.

fp 자료형의 핵심 함수는 `\fp_eval:n`이다. 인자로 주어지는 식을 parsing하여 결과를 fp형식으로 처리한다.

```
\ExplSyntaxOn
\fp_eval:n { 10/3 }
\ExplSyntaxOff
```

3.333333333333333

중요한 연산자와 부호는 다음과 같다.

- +, -, *, /, ( )

- `e` 부동소수점 실수 표현에 사용하는 부호이며 자연로그의 밑 e와는 관계없다. `1.666e2` = 166.6
- `inf`무한대.$+\infty$.
- `**`지수연산자.`^`를써도된다고하나`**`를일관되게쓰는것을권장한다.`\fp_eval:n {2**10}`=1024
- `&&`, `||`, `!` 논리 연산자 and, or, not이다. 다른 자료형에서와 달리 다음과 같은 경우도 의미를 가진다. 이 연산의 결과가 0인 것은 boolean FALSE인 것과 동일하다. 1이면 TRUE이다.

```
\ExplSyntaxOn
\fp_set:Nn \l_tmpa_fp { 2.0 }
\fp_eval:n { \l_tmpa_fp < 0.5 || \l_tmpa_fp = 2.0 }
\ExplSyntaxOff
```

1

- `?:` 3항 연산자이다. `\fp_eval:n { A ? a : B ? b : c }` 형식으로 쓸 수 있다. if A then a elseif B then b else c fi fi의 의미를 갖는다. A, B, C는 조건 연산식이고 a, b, c는 fp 값이다. (즉 일반 매크로가 a, b, c 자리에 올 수 없다.) C같은 언어에 익숙하다면 이 3항 연산자도 쉽게 이해할 수 있겠지만, 코드의 가독성이 떨어지기 때문에 남용하는 것은 권하지 않는다.

```
\ExplSyntaxOn
\fp_set:Nn \l_tmpa_fp { 2.00 }
\fp_eval:n {
    \l_tmpa_fp < 0.5 ? 1.0 :
    \l_tmpa_fp ** 2 < 3 ? 2.0 : 3.0
}
\ExplSyntaxOff
```

3

fp 함수는 `\fp_eval:n` 범위 안에서 연산할 수 있는 함수들이다.

- `exp, ln, fact` : $e^{\cdot}$, ln, factorial. 자연로그의 밑 (e)을 나타내려면

```
\ExplSyntaxOn
\fp_set:Nn \l_e_fp { exp(1) }
\fp_use:N \l_e_fp
\ExplSyntaxOff
```

2.718281828459045

```
\ExplSyntaxOn
$\log \c_math_subscript_token {10} 2 = \fp_eval:n { ln(2) / ln(10) } $
\ExplSyntaxOff
```

$\log_{10} 2 = 0.3010299956639811$

- `sin, cos, tan, cot, csc, sec` : 호도법 (radian) 각을 인자로 취한다.

- `sind, cosd, tand, cotd, cscd, secd` : 도(degree)를 인자로 취한다.
- `sqrt` 제곱근.
- `pi` : $\pi \simeq 3.141592653589793$.
- `round, trunc` 반올림 또는 버림.

```
\ExplSyntaxOn
\fp_eval:n { round ( 3.14159 ) }
    \par
\fp_eval:n { round ( 3.14159, 3 )
    }
\ExplSyntaxOff
```

```
3
3.142
```

- `ceil, floor`

type 변환

① 숫자로 이루어진 tl은 확장하여 `\fp_eval:n` 하면 된다. `\fp_eval:n`의 인자 영역에 들어간 한 번 또는 두 번 확장가능한 tl은 자연스럽게 확장된다.

② 정수(int)는 `\fp_eval:n` 범위 안에서 바로 쓸 수 있다.

```
\ExplSyntaxOn
\int_set:Nn \l_tmpa_int { 2 }
\fp_eval:n { sin ( ( \l_tmpa_int ** 0.4 ) pi ) }
\ExplSyntaxOff
```

```
-0.8434985587189794
```

③ `\fp_eval:n`의 결과의 소수부(decimal part)가 0이면 int에 자연스럽게 할당된다.

④ `floor, ceil` 연산의 결과는 int에 할당된다.

⑤ 일반적인 fp는 `trunc 0` 하거나 `round 0`하면 int에 할당된다.

```
\ExplSyntaxOn
\int_set:Nn \l_tmpa_int {
    \fp_eval:n { 6/3 } }
\int_use:N \l_tmpa_int \par
\int_set:Nn \l_tmpa_int {
    \fp_eval:n { ceil ( 3/2 ) } }
\int_use:N \l_tmpa_int \par
\int_set:Nn \l_tmpa_int {
    \fp_eval:n { trunc ( 100 *
    sind ( 20 ) ) } }
\int_use:N \l_tmpa_int
\ExplSyntaxOff
```

```
2
2
34
```

길이(dim)와 fp의 관계는 아주 중요하므로 dim을 다루는 곳에서 자세히 연습하기로 한다.

이제 다음 문제를 풀어보자.

보조문제 1

주어진 정수의 제곱근을 넘지 않는 최대 정수를 출력하여라.

x를 넘지 않는 최대 정수를 $\lfloor x \rfloor$로 나타내기로 하면,

```
\ExplSyntaxOn
$\lfloor 2.95 \rfloor = \fp_eval:n {
    floor ( 2.95 ) } $
\ExplSyntaxOff
```

$\lfloor 2.95 \rfloor = 2$

이다. 또는 인자 없이 trunc하여도 같다. 이 때 truncate가 소수점 위치에서 일어나기 때문이다.

```
\ExplSyntaxOn

\cs_new:Npn \get_floor_sqrt:n #1
{
    \fp_eval:n { floor ( sqrt ( #1 ) ) }
}

\get_floor_sqrt:n { 20 }

\ExplSyntaxOff
```

```
4
```

2 cs와 인자 확장

우리는 지금까지 함수를 정의하는 데 `\cs_new:Npn`을 사용해왔다. 이제 `\cs_set:Npn`을 소개하려 하는데, 이 명령은 몇 가지 용법을 가지고 있다.

(1) 이미 정의된 함수(cs)의 내용을 수정할 때. 실제로는 `\cs_new:Npn` 하지 않아도 바로 `\cs_set:Npn`할 수 있는데 그렇게 하지 않도록 유도하는 이유는 plain TEX의 `\def` 위험성과 같은 이유에서이다. 즉 그 함수가 이미 정의되어 있는지를 체크하지 않기 때문에 발생하는 위험을 줄이기 위해서 `\cs_new`를 쓰도록 한 것이다.

(2) 함수(즉 :와 인자형 지시자를 반드시 갖는 cs) 이름만이 아니라 일반 매크로의 내용을 정의하려 할 때

(3) 지역적으로(locally) 함수의 동작을 잠시 바꾸려 할 때. 그룹 안에서 `\cs_set:N`한 것은 그룹을 벗어나면 효력을 잃는다.

다음에 예를 들어본다.

```
\ExplSyntaxOn
\cs_new:Npn \foo_fn:n #1 {
    This~is~#1 }
\foo_fn:n { test }
\par
\cs_set:Npn \foo_fn:n #1 {
    You~are~#1 }
\foo_fn:n { beautiful }
\ExplSyntaxOff
```

This is test
You are beautiful

```
\ExplSyntaxOn
\cs_set:Npn \mytest #1 { Hello,~#1!
    }
\mytest{boys}
\ExplSyntaxOff
```

Hello, boys!

이 두 번째 사용법을 이용하면 \NewDocumentCommand를 쓰지 않더라도 문서 명령을 만들 수 있다. 즉

```
\ExplSyntaxOn
\cs_set_eq:NN \hello \foo_fn:n
\ExplSyntaxOff
\hello{nice}
```

This is nice

이와 같이 정의할 수 있는 것이다.
일반적인 LaTeX 명령은 "풀리지 않게" 정의하는 것이 책갈피나 moving arguments에서 깨지지 않게 보호할 수 있다. 그렇기 때문에 사용자 문서 명령은 \NewDocumentCommand를 쓰는 것이 바람직하다. 그렇지만 아주 특별히, 풀리는 명령을 정의해야 할 필요가 있을 때, 그것도 문서 명령 비슷하게 해야 할 때 이것이 한 가지 팁이 될 수 있다. (그러나 \cs_if_... 명령으로 이렇게 정의된 명령을 검사하면 에러가 발생할 가능성도 있으므로 주의를 요한다.)

cs를 정의할 때 인자를 확장하려면 어떻게 하는가? 예를 들어 \foo_fn:o라는 형태로, 즉 들어오는 인자를 무조건 한 번 확장하여 사용하려 한다면 다음과 같이 정의한다.

```
\ExplSyntaxOn
\cs_new:Npn \foo_a:n #1
{
    \fbox{#1}
}

\cs_generate_variant:Nn \foo_a:n { V, o }
\ExplSyntaxOff
```

\cs_generate_variant:Nn은 모든 확장형을 다 쓸 수 있게 해주는 것은 아니다. 여기서는 다음 사항을 기억하자.

(1) 인자가 tl일 때, n을 V, o, x로 확장할 수 있고, N을 V로 확장할 수 있다.

(2) 인자가 int, fp, dim일 때, n을 o, x로 확장할 수 있고, N을 V로 확장할 수 있다.

유념할 것은 예컨대 자신이 함수의 이름을 `\my_func:x`라고 지었다고 해서 바로 x확장이 일어나는 것은 아니라는 것이다. 그러므로 일단 모든 인자를 무조건 n으로 하는 함수 기본 형을 정의한 후에 이의 variant를 기술하는 것이 좋은 코딩 방법이 된다.

3 소수 구하기

원래의 문제로 돌아와서, 주어진 알고리즘을 잘 상고해보면 `break` 문이 들어 있는 것을 볼 수 있다. 그러므로 for 루프를 쓰도록 되어 있지만 우리는 map을 활용해야 한다.
n이 주어지면 그 수까지의 자연수를 담고 있는 clist를 먼저 만들기로 하자. 1은 제외한다.

```
\ExplSyntaxOn

\cs_new:Npn \build_numlist:n #1
{
    \clist_clear:N \l_tmpa_clist
    \int_step_inline:nnn { 2 } { #1 }
    {
        \clist_put_right:Nn \l_tmpa_clist { ##1 }
    }
}

\build_numlist:n { 100 }
\clist_use:Nn \l_tmpa_clist { , ~ }

\ExplSyntaxOff
```

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100

그리고, 주어진 수의 $\lfloor x \rfloor$까지의 clist도 하나 만든다.

```
\ExplSyntaxOn

\cs_set:Npn \build_numlist:n #1
{
    \clist_clear:N \l_tmpa_clist
    \int_step_inline:nnn { 2 } { #1 }
    {
        \clist_put_right:Nn \l_tmpa_clist { ##1 }
    }
```

```
    \clist_clear:N \l_tmpb_clist
    \int_step_inline:nnn { 2 } { \get_floor_sqrt:n { #1 } }
    {
        \clist_put_right:Nn \l_tmpb_clist { ##1 }
    }
}

\build_numlist:n { 100 }
a~:~\clist_use:Nn \l_tmpa_clist { , ~ } \par
b~:~\clist_use:Nn \l_tmpb_clist { , ~ }

\ExplSyntaxOff
```

a : 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100
b : 2, 3, 4, 5, 6, 7, 8, 9, 10

처음에 제시한 알고리즘을 구현하면 다음과 같다.

```
\ExplSyntaxOn
\cs_set:Npn \get_floor_sqrt:n #1
{
    \fp_eval:n { floor ( sqrt ( #1 ) ) }
}

\cs_set:Npn \build_numlist:n #1
{
    \clist_clear:N \l_tmpa_clist
    \int_step_inline:nnn { 2 } { #1 }
    {
        \clist_put_right:Nn \l_tmpa_clist { ##1 }
    }

    \clist_clear:N \l_tmpb_clist
    \int_step_inline:nnn { 2 } { \get_floor_sqrt:n { #1 } }
    {
        \clist_put_right:Nn \l_tmpb_clist { ##1 }
    }
}

\cs_new:Npn \fn_primes:n #1
{
    \clist_gclear:N \g_tmpa_clist
```

```
    \build_numlist:n { #1 }

    \clist_map_inline:Nn \l_tmpa_clist
    {
        \bool_gset_true:N \g_tmpa_bool
        \int_gset:Nn \g_tmpa_int { ##1 }
        \clist_map_function:NN \l_tmpb_clist \sub_fn:n
        \bool_if:NT \g_tmpa_bool
        {
            \clist_put_right:Nn \g_tmpa_clist { ##1 }
        }
    }

    \clist_use:Nn \g_tmpa_clist { , ~ }
}

\cs_new:Npn \sub_fn:n #1
{
    \bool_if:nT
    {
        \int_compare_p:n { \int_mod:nn { \g_tmpa_int } { #1 } == 0  }
        &&
        \int_compare_p:n { \g_tmpa_int != #1 }
    }
    {
        \bool_gset_false:N \g_tmpa_bool
        \clist_map_break:
    }
}

\cs_set:Npn \fnprimes #1
{
    \fn_primes:n { #1 }
    {}~( \clist_count:N \g_tmpa_clist )
}

\ExplSyntaxOff

\fnprimes{21}
```

2, 3, 5, 7, 11, 13, 17, 19 (8)

연습문제

기본 1. 본문의 예제는 루프를 탈출하기 위하여 `\clist_map`을 활용하였다. 그런데 while do를 쓰면 clist mapping을 이용하지 않아도 루프의 탈출 조건을 만들 수 있다. 이를 이용하여 같은 알고리즘을 구현할 수 있겠는가?

발전 2. 주어진 수가 소수인지를 검사하는 다음과 같은 알고리즘이 있다. 이를 expl3로 구현할 수 있겠는가?

```
>>> def isPrime(n):
        if (n<=1):
                return False
        if (n<=3):
                return True
        if (n%2 == 0 or n%3 == 0):
                return False
        i=5
        while (i*i <= n ):
                if (n%i == 0 or n%(i+2) == 0):
                        return False
                i = i+6
        return True

>>> if (isPrime(43)):
        print('prime')
else:
        print('not a prime')


prime
```

실력 3. KTUG 게시판 :235888 글에 소인수분해 알고리즘이 소개되어 있다. 이를 바탕으로 다음 순서로 문제를 해결하여라.

① 두 수를 인자로 받아서 최대공약수를 구하여라.

② 최대공약수를 소인수분해하여 결과를 clist나 seq에 저장하여라.

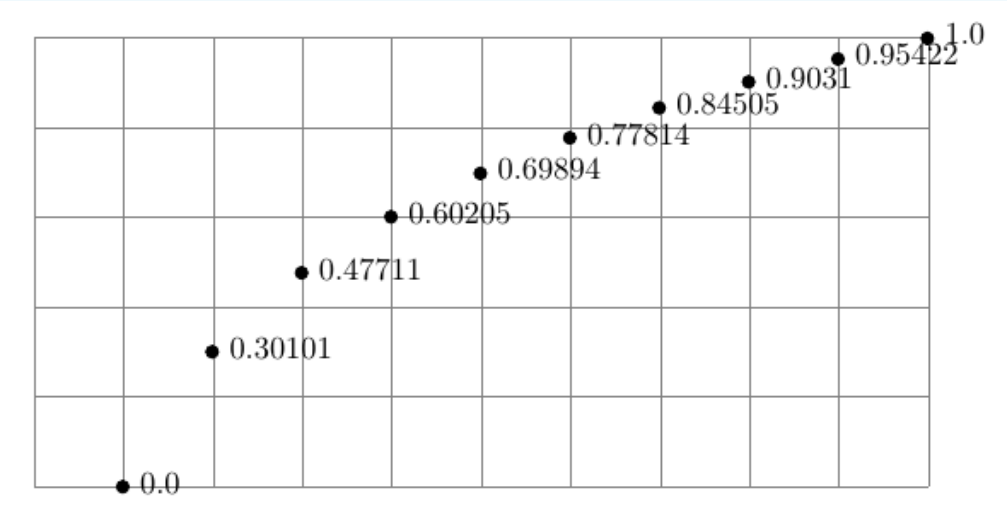
③ 최대공약수의 소인수를 취하여 차례로 두 수를 나누어가면서 몫(quotient)의 변화 과정을 clist나 seq에 저장하여라.

④ 준비된 세 개의 clist (seq)를 이용하여 다음 그림과 같이 출력하여라.

2	16	24
2	8	12
2	4	6
	2	3

문제

1부터 10까지 정수의 상용로그 값을 계산하여 좌표평면에 다음 그림과 같이 나타내어라.



먼저, 1부터 10까지 정수의 상용로그 값을 도표로 작성해보자.

```
\ExplSyntaxOn

\cs_new:Npn \format_num:n #1
{
    \tl_set:Nn \l_tmpa_tl { #1 }
    \str_if_in:NnF \l_tmpa_tl { . }
    {
        \tl_put_right:Nn \l_tmpa_tl { .0000 }
    }

    \int_compare:nT { \tl_count:N \l_tmpa_tl < 6 }
    {
        \int_step_inline:nn { 6 - \tl_count:N \l_tmpa_tl }
        {
            \tl_put_right:Nn \l_tmpa_tl { 0 }
        }
    }

    \l_tmpa_tl
}

\cs_generate_variant:Nn \format_num:n { V, x }

\cs_new:Npn \print_log_value:n #1
{
    \ensuremath{#1} \c_alignment_token
    \ensuremath{
```

```
        \format_num:x { \fp_eval:n { round ( ln (#1) / ln (10), 4 ) } }
    }
    \tabularnewline \hline
}

\begin{tabular}{|r|c|}
\hline
$x$ & $y$ \\ \hline
\int_step_inline:nn { 9 }
{
    \print_log_value:n { #1 }
}
\ensuremath{10} \c_alignment_token
\ensuremath{
    \format_num:n { 1 } }
\\ \hline
\end{tabular}
\ExplSyntaxOff
```

x	y
1	0.0000
2	0.3010
3	0.4771
4	0.6021
5	0.6990
6	0.7782
7	0.8451
8	0.9031
9	0.9542
10	1.0000

4 Ti*k*Z와 expl3

현 시점에서 "그리기"의 사실상 표준인 Ti*k*Z를 expl3 syntax 범위 안에서 쓰기 위해서 다음 두 가지를 주의하여야 한다.

(1) Ti*k*Z 옵션 등에 나타나는 space를 반드시 명시적으로 (틸데로써) 입력하여야 한다.
`\tikz[rounded~corners=3pt]`

(2) 콜론 문자를 직접 입력하여서는 안 된다. 이럴 때를 위하여 `\c_colon_str`이라는 string 상수가 정의되어 있다.

(3) Ti*k*Z 환경과 expl3는 서로 다른 확장 규칙을 가지고 있다. expl3로 작성된 함수가 Ti*k*Z 환경 안에서 성공적으로 풀리지 않을 수 있다. 이럴 때에는 Ti*k*Z 환경을 ExplSyntax 범위 밖에 두는 것을 고려해보아야 한다.

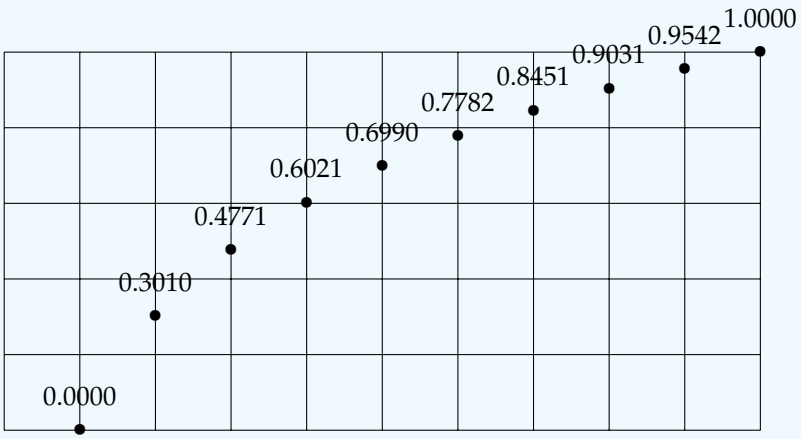
(4) Ti*k*Z 환경 내부에서 반복문을 실행해야 한다면 `\foreach`를 쓰는 것이 대체로 안전하다.

(5) 될 수 있으면 expl3로 행하는 계산이나 함수의 확장은 Ti*k*Z 환경 외부에서 실행하고 결과를 expandable 한 tl이나 매크로로 Ti*k*Z 함수에 넘겨주도록 코드를 작성하는 것이 좋다.

이 몇 가지를 주의하면 ExplSyntax 안에서 Ti*k*Z를 사용하는 것이 가능하다.

```
\ExplSyntaxOn
\cs_new:Npn \mylog:n #1
{
    \fp_eval:n { round ( ln ( #1 ) / ln ( 10 ) , 4 ) }
}
\cs_set_eq:NN \mylog \mylog:n

\begin{tikzpicture}
\draw (0,0) grid (10,5);
\foreach \x in {1,2,...,10}
    \node at (\x, 5*\mylog{\x}) [label={\format_num:x { \mylog{\x}} }]
    {$\bullet$};
\end{tikzpicture}
\ExplSyntaxOff
```



확장 (4) `\exp_args:` 함수는 그 이름에서 알 수 있는 바와 같이 arguments를 expand하는 데 쓰는 것이다. 그런데 arguments가 아니라 멀쩡한 매크로가 있는 자리에서 그 매크로 자체를 (`\expandafter` 처럼) 확장하고자 한다면 어떻게 해야 하는가?
이럴 때 쓰는 것이 `\exp_last_unbraced:` 함수이다. 예를 들어보자.

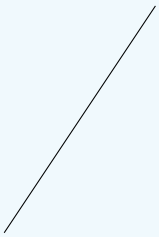
```
\ExplSyntaxOn
\begin{tikzpicture}
\draw (0,0) -- (3,3);
\end{tikzpicture}
\ExplSyntaxOff
```

이 경우에, 선분의 끝점을 매크로로 전달하는 상황을 가정한다.

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl { [very~thick,blue] }
\cs_set:Npn \my_point:n #1 { (#1,\int_eval:n { #1+1 } ) }
\tl_set:Nx \l_tmpb_tl { (0,0) -- \my_point:n { 2 } }
\begin{tikzpicture}
\draw \l_tmpa_tl \l_tmpb_tl ;
\end{tikzpicture}
\ExplSyntaxOff
```

(이 샘플은 별도로 확장 안 해도 잘 동작하지만 설명을 위해 예를 드는 것이므로) 여기서 `\l_tmpa_tl`와 `\l_tmpb_tl` 매크로를 확장하여야 할 적에 다음과 같이 한다.

```
\ExplSyntaxOn
\tl_set:Nn \l_tmpa_tl { [very~thick,blue] }
\cs_set:Npn \my_point:n #1 { (#1,\int_eval:n { #1+1 } ) }
\tl_set:Nx \l_tmpb_tl { (0,0) -- \my_point:n { 2 } }
\begin{tikzpicture}
\exp_last_unbraced:Noo \draw \l_tmpa_tl \l_tmpb_tl ;
\end{tikzpicture}
\ExplSyntaxOff
```



요약하면, 브레이스(`{}`)로 둘러싸여 전달되는 부분을 확장하려면 `\exp_args:`를 쓰고 단일 매크로를 확장하려 할 때는 `\exp_last_unbraced:`를 쓴다고 해도 좋다.
이미 강조한 바이지만, 예를 들어

```
\my_func:n { \l_tmpa_tl }
```

이런 상황에서 `\l_tmpa_tl`이 단 한 개의 문자로 이루어져 있을 때 `\my_func:n \l_tmpa_tl`라고 쓰고 싶은 유혹을 충분히 느끼겠지만 확장 관련 문제가 복잡하게 얽힐 때에 대비하여 n 아규멘트라면 중괄호를 꼭 써주어야 한다.

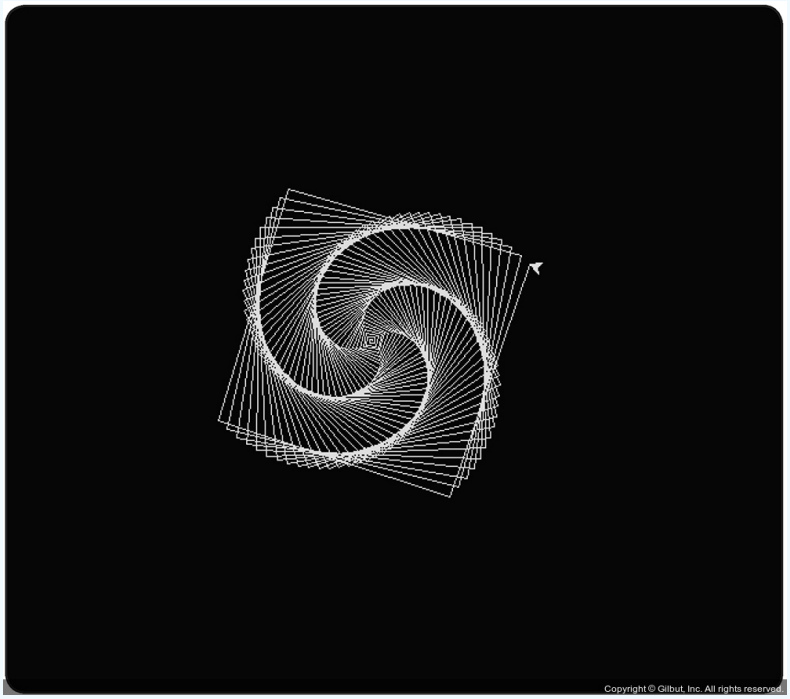
연습문제

기본 1. 가로 10cm, 세로 10cm이고 상하좌우 여백이 1.5cm, 모두 30페이지를 가진 pdf 문서를 작성한다. 매 페이지마다 현재 페이지가 전체 페이지수에 대하여 몇 %인지를 표시하고 페이지 번호가 5의 배수가 되는 때 페이지의 배면색상(background color)를 cyan으로 하여라.

기본 2. 위의 pdf 문서 각 페이지의 중앙에 반지름 3cm인 원을 그리고 진행비율(현재페이지/전체페이지)을 붉은 색으로 표시하는 progress pie를 그려라.

기본 3. 중학교 수학 교과서의 부록으로 "삼각비표"가 있다. 이 표의 일부 (0° 부터 25° 까지)를 되도록 예쁘게 작성하여라.

실력 4. 다음 그림을 그려보아라. 배경색은 별도로 지정하지 않아도 좋다.



Nova de Hi.