

Basic Math

By 윤명근 / 박수현

수업목표

- Four basic operations
- 대입연산자 (할당 연산자 : assignment operator)
- Order of operations
- bit 연산자
- More operators
- Really big and really small
- 진법변환
- 심화학습

Four basic operations

- 숫자

- Python은 크게 정수(int), 실수(float) 그리고 복소수(complex) 형태의 숫자를 표현할 수 있음

자료형	설명	예
int	정수를 표현하는 자료형	123
float	실수를 표현하는 자료형	1.43
complex	복소수를 표현하는 자료형	5+4i

Four basic operations

- 사칙연산(operations)과 연산자(operator)
 - 덧셈(+), 뺄셈(-), 곱셈(*), 나눗셈(/)

```
>>> print (3+4)
7
>>> print (8-5)
3
>>> print(6/2)
3.0
>>> print (6//2)
3
>>> print(6%2)
0
```

Four basic operations

- 나누기 연산
 - 몫: //
 - 나머지(Modular): %

```
>>> print(7//2)
3
>>> print(7%2)
1
```

대입 연산자

- '='는 할당 연산자(assignment operator)

```
>>>  
>>> happy_hour = "12시"  
>>> print("- 즐거운 시간 :", happy_hour)  
- 즐거운 시간 : 12시  
>>>
```

대입 연산자

- '='는 할당 연산자(assignment operator)
- 동일함을 나타내는 '=='과 구분할 줄 알아야 함

```
assign-equal.py -
File Edit Format Run Options Window Help
happy_time = 12

if happy_time == 12:
    print("- 12시는 점심시간, 행복한 시간")
else:
    print("- 12시까지 언제 기다려. 배고파")
...

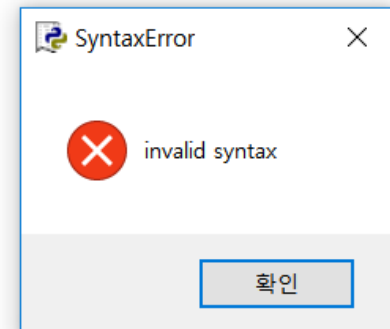
if happy_time = 12:
    print("- 12시는 점심시간, 행복한 시간")
else:
    print("- 12시까지 언제 기다려. 배고파")
...
```

- 12시는 점심시간, 행복한 시간

```
assign-equal.py -
File Edit Format Run Options Window Help
happy_time = 12

if happy_time == 12:
    print("- 12시는 점심시간, 행복한 시간")
else:
    print("- 12시까지 언제 기다려. 배고파")

if happy_time = 12:
    print("- 12시는 점심시간, 행복한 시간")
else:
    print("- 12시까지 언제 기다려. 배고파")
```



대입 연산자

- 다양한 축약 형태 지원

`+=, -=, *=, /=, **/, //=, %=`

Ex) `x += 10` → `x = x + 10`

`x -= 10` → `x = x - 10`

`x *= 10` → `x = x * 10`

`x /= 10` → `x = x / 10`

Order of Operations

- 사칙연산 우선 순위 유지
- 괄호
- 기타 : *, /, %
+, -
- 줄여 쓰기
 - `number +=1` → `number = number + 1`
 - `number /=2` → `number = number / 2`

```
>>>
>>> print (2+4*3)
14
>>> print ( (2+4) *3)
18
>>>
```

Order of Operations

- 실습
 - 무리수 e를 구하는 코드를 작성하고, 실제 값과 비교해보자.

$$\lim_{x \rightarrow \infty} \left(1 + \frac{1}{x}\right)^x = e$$

irrational_number.py

File Edit Format Run Options Window Help

수학과 관련된 모듈

import math

```
print(" e 실제 값 :", math.e)
print("x = 10      :", (1 + 1/10) ** 10)
print("x = 100     :", (1 + 1/100) ** 100)
print("x = 1,000    :", (1 + 1/1000) ** 1000)
print("x = 10,000   :", (1 + 1/10000) ** 10000)
print("x = 100,000  :", (1 + 1/100000) ** 100000)

print("x = 100,000,000 :", (1 + 1/100000000) ** 100000000)
```

```
e 실제 값 : 2.718281828459045
x = 10      : 2.5937424601000023
x = 100     : 2.7048138294215285
x = 1,000   : 2.7169239322355936
x = 10,000  : 2.7181459268249255
x = 100,000 : 2.7182682371922975
x = 100,000,000 : 2.7182817983473577
>>>
```

bit 연산자

- Bitwise operator
- Computer는 정보를 bit로 표시
 - 8 bit → 1 byte, 32/64bit → 1워드(word)
- Bit(이진 데이터 : binary data) level의 연산자
- Bit 논리 연산자 (logical operator)
- Bit shift 연산자
- Bit mask

bit 연산자

- bit 논리 연산자

& : bit 논리곱

| : bit 논리합

^ : bit XOR

- ^ : 캐럿(carrot)

- XOR : Exclusive OR

~ : bit 논리 부정

A	B	A&B	A B	A^B	~A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Bit 연산자

- bit 논리 연산자

```
# 비트 연산자
# & 는 bit단위에서 같을 때 1, 다를 때 0 이다.
print("23 & 5 :", 23 & 5)
# | 은 bit단위에서 한개라도 1이면 1, 아니면 0이다.
print("23 | 5 :", 23 | 5)
# ^ 은 bit단위에서 두 bit가 다르면 1, 같으면 0이다.
print("23 ^ 5 :", 23 ^ 5)
# a << b 를 하면 a 를 b번만큼 왼쪽으로 bit이동을 시킨다.
print("3 << 3 :", 3 << 3)
# a >> b 를 하면 a 를 b번만큼 오른쪽으로 bit이동을 시킨다.
print("32 >> 2 :", 32 >> 2)
```

File Edit Format Run Options Window Help

```
a = 16 + 4 + 2 + 1 #23 0b 0001 0111
b = 4 + 1           #5  0b 0000 0101
```

```
print()
print("a = ", bin(a))
print("b = ", bin(b))
```

```
print()
print("a & b=", a & b, "Binary =", bin(a & b))
print("a | b=", a | b, "Binary =", bin(a | b))
print("a ^ b=", a ^ b, "Binary =", bin(a ^ b))
print("~b=", ~b, "Binary =", bin(~b & 0xFF))
```

a	0001 0111
b	0000 0101
a & b	0000 0101
a b	0001 0111
a ^ b	0001 0010
~b	1111 1010

```
a = 0b10111
b = 0b101
```

```
a & b= 5 Binary = 0b101
a | b= 23 Binary = 0b10111
a ^ b= 18 Binary = 0b10010
~b= -6 Binary = 0b11111010
```

bit 연산자

- bit 논리 연산자

- ^ (XOR) 연산자 활용 사례 → 암호

- a와 b가 어떤 값이라도 XOR 연산에 a가 두 번 등장하면 b만 남게 됨

- $a \wedge b \wedge a = b$

- $0 \wedge 0 \wedge 0 = 0$

- $0 \wedge 1 \wedge 0 = 1$

- $1 \wedge 0 \wedge 1 = 0$

- $1 \wedge 1 \wedge 1 = 1$

- $(a \wedge b)$ 를 안다고 하더라도, a를 모르면 b를 알아낼 수 없으며, b를 모르면 a를 알아낼 수 없음

- b: 비밀을 유지하며 친구에게 전달할 문자

- a: 친구와 사전에 약속한 비밀문자

- 친구에게 $c(a \wedge b)$ 전달

- 친구는 c와 a를 XOR 연산해서 b를 알아냄

- 다른 사람들은 c를 안다고 하여도 a와 b를 알아낼 수 없음

bit 연산자

- bit 쉬프트(shift) 연산자
 - \ll : 왼쪽으로 bit 이동
 - $1 \ll 3$: 숫자 '1'의 모든 bit를 왼쪽으로 3번 이동시킴. \rightarrow 8로 변환
 - 숫자를 n 번 왼쪽으로 이동시키면 2^n 을 곱하는 효과와 동일해짐
 - \gg : 오른쪽으로 bit 이동
 - $8 \gg 3$: 숫자 '8'의 모든 bit를 오른쪽으로 3번 이동시킴. \rightarrow 1로 변환
 - 숫자를 n 번 오른쪽으로 이동시키면 2^n 으로 나누는 효과와 동일해짐

2.2example3.py

File Edit Format Run Options Window Help

```
print()
print( "1<<3 :", 1<<3) # 1 : 0000 0001 --> 8 : 0000 1000
print( "8>>3 :", 8>>3) # 8 : 0000 1000 --> 1 : 0000 0001

print()
print( "11>>1 :", 11>>1) # 11 : 0000 1011 --> 5 : 0000 0101
print( "11>>2 :", 11>>2) # 11 : 0000 1011 --> 2 : 0000 0010
```

$1 \ll 3 : 8$
 $8 \gg 3 : 1$

$11 \gg 1 : 5$
 $11 \gg 2 : 2$

bit 연산자

- bit mask
 - 특정 위치의 bit 값을 알고 싶을 때 사용 (0 or 1?)
 - x의 오른쪽에서부터 n번째 bit 값 확인
 - $1 \& (x \gg n-1)$
 - 연산 후에도 x값은 변화가 없음

```
2.2example4.py
File Edit Format Run Options Window Help
1 x = 1024 # 0000 00100 0000 0000
2 print()
3 print ( ">> 11th bit of %d :" %x, 1 & (x>>10))
4
5 x = 4 # 0000 0100
6 print()
7 print ( ">> 1st bit of %d :" %x, 1 & (x>>0))
8 print ( " 2nd bit of %d :" %x, 1 & (x>>1))
9 print ( " 3rd bit of %d :" %x, 1 & (x>>2))
10
```

>> 11th bit of 1024 : 1

>> 1st bit of 4 : 0
2nd bit of 4 : 0
3rd bit of 4 : 1

>>>

More Operators

- 지수법
 - 제공하기
 - **
 - $2^{**}3=8$

```
>>>
>>> 2**3
8
>>> 2**10
1024
>>> 2**100
1267650600228229401496703205376
>>>
```

Really Big and Really Small

- E-표기법(Exponential-notation)
 - 소수와 10의 제곱승 이용
 - e+08은 10의 8승
 - e-04은 10의 -4승

```
>>> x = 987654321.0
>>> print(x)
987654321.0
>>> print("%e" %x)
9.876543e+08
>>>
>>> y = 0.000123
>>> print(y)
0.000123
>>> print("%e" %y)
1.230000e-04
```

진법 변환

- 10진수 -> 2진수 변환

2	2017		
2	1008	...	1
2	504	...	0
2	252	...	0
2	126	...	0
2	63	...	0
2	31	...	1
2	15	...	1
2	7	...	1
2	3	...	1
	1	...	1

Computer에서 2진수를 사용하는 이유?

$$2017_{(10)} = 11111100001_{(2)}$$

진법 변환

- 2진수 → 16진수 변환
 - 4 bit의 숫자를 하나로 변환
 - $11111100001_{(2)} = 7E1_{(16)}$
→ 0111 1110 0001
- 2진수 → 8진수 변환
 - 3bit의 숫자를 하나로 변환
 - $11111100001_{(2)} = 3741_{(8)}$
→ 011 111 100 001

10진수	2진수	8진수	16진수
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

진법 변환

- 2진수 덧셈

	0111	7
+	0011	3
	1010	10

- 2진수 뺄셈

	0111	7
?	0011	3
	?	4

– 7 – 3이 아닌 7 + (-3)으로 처리

진법 변환

- 음의 10진수 -> 2진수 변환 (MSB 이용)
 - MSB(Most Significant Bit) : 가장 큰 자릿수의 bit
 - 일반적으로 가장 왼쪽 bit
 - MSB가 0일 때 양수, 1일 때 음수
$$123_{(10)} = 0111\ 1011_{(2)} \quad (= 7B_{(16)})$$
$$-123_{(10)} = 1111\ 1011_{(2)}$$

- 문제점 : 1010 (십진수 10) 이 0을 표시 ?

	0001	1
+	1001	-1
	1010	0
	(→ 십진수 10)	

- 1의 보수(1's complement)를 사용해서 해결하자

진법 변환

- 음의 10진수 -> 2진수 변환 (**1의 보수 사용**)

- 1을 0으로 0을 1로 바꿔 줌.

$$123_{(10)} = 0111\ 1011_{(2)} (= 7B_{(16)})$$

$$-123_{(10)} = 1000\ 0100_{(2)}$$

- 또 다른 문제발생

	0001	1
+	1110	-1
	1111(-0)	0

- 0의 표현이 2가지($0000\ 0000_{(2)}(+0)$ 과 $1111\ 1111_{(2)}(-0)$)

- 1의 보수 대신 2의 보수 (2's complement) 를 사용하자**

진법 변환

- 음의 10진수 -> 2진수 변환(2의 보수 사용)

- 1의 보수 후 1을 더해줌

$$123_{(10)} = 0111\ 1011_{(2)}$$

$$-123_{(10)} = 1000\ 0101_{(2)}$$

- 해결

	0001	1
+	1111	-1
	0000	0

- (8bit 기준) -128 ~ 127까지 표현 가능

$$1000\ 0000_{(2)} \sim 0111\ 1111_{(2)}$$

진법 변환

- 표현 가능한 범위를 넘으면?

싸이의 세계적인 히트곡 '강남스타일'은 2012년 7월 발표된 이후 현재까지 유튜브 조회수가 21억 건을 돌파하면서 최다 조회수를 기록했다. 이 같은 성과는 경이적이다. 1일(현지 시각) 유튜브는 모회사 구글의 SNS(소셜네트워크서비스)인 구글플러스를 통해 강남스타일 조회수가 집계 한계치를 넘어서 집계 시스템을 '업그레이드'해야 했다고 밝혔다. 그러면서 처음에 유튜브가 설계됐을 때 조회수가 21억4,748만3,647건(유튜브 운영 시스템인 '32 bit 정수'로 표현할 수 있는 최대 조회수)을 넘어서는 동영상의 수가 있을 거라고는 전혀 예상하지 못했다고 덧붙였다. 유튜브의 맷 맥러논은 회사는 조회수 20억 건이 충분할 것으로 생각했지만, 그렇지 않았다고 언급했다.



<http://kr.wsj.com/posts/2014/12/04/%EC%8B%B8%EC%9D%B4-%EA%B0%95%EB%82%A8%EC%8A%A4%ED%83%80%EC%9D%BC-%EB%95%8C%EB%AC%B8%EC%97%90-%EC%9C%A0%ED%88%AC%EB%B8%8C-%EC%A7%91%EA%B3%84-%EC%8B%9C%EC%8A%A4%ED%85%9C-%EC%97%85%EA%B7%B8%EB%A0%88/>

싸이의 '강남스타일' 인기는 상상을 초월합니다. 적어도 유튜브 코드에 의하면 확실히 그랬죠. 유튜브 조회수가 2,147,483,647 (21억) 을 넘어가면서 조회수를 64bit 숫자로 바꾸어 9,223,372,036,854,775,808(922경)까지 평가할 수 있게 바꾸었죠. 유튜브는 왜 조회수를 21억 이상 셀 수 없었던 걸까요?

-중략-

여기서, 유튜브의 가능한 조회수가 2,147,483,647로 4,294,967,295가 아니라는 데 주목할 필요가 있습니다. 이건 왜 다른 걸까요? 32bit가 양수가 아니기 때문입니다. 음수를 표기하기 위해서는 절반인 2,147,483,647을 음수에 할당한 것이죠. 유튜브의 32 bit 숫자는 0에서 4,294,967,295까지 표기하는 대신 -2,147,483,648에서 2,147,483,647까지 표기합니다. 유튜브의 조회는 음수가 될 일이 없지만 일반 데이터베이스나 프로그래밍을 할 때는 음수를 써야 될 때가 있습니다. 따라서 첫 자리는 음수인지 양수인지 판별하는 부호로 읽고 32bit 숫자는 -2,147,483,648로 2,147,483,647 읽는 것이 Computer의 표준 언어지요. 유튜브의 조회수가 20억을 넘어가면서 32bit가 표현할 수 있는 최고 숫자를 넘어가자 더 이상 Computer는 제대로 숫자를 읽을 수가 없었습니다. 결국 유튜브는 조회수를 64bit로 만들고 9,223,372,036,854,775,808(922경)까지 읽게 변환하였죠. (<http://newspeppermint.com/2014/12/16/binary-bug/>)

진법 변환

- 10진수 -> 2진수 변환 : `bin(int)`

`bin(x)`

Convert an integer number to a binary string. The result is a valid Python expression. If `x` is not a Python `int` object, it has to define an `__index__()` method that returns an integer.

- 10진수 -> 8진수 변환 : `oct(int)`

`oct(x)`

Convert an integer number to an octal string. The result is a valid Python expression. If `x` is not a Python `int` object, it has to define an `__index__()` method that returns an integer.

진법 변환

- 10진수 -> 16진수 변환 : hex(int)

hex(x)

Convert an integer number to a lowercase hexadecimal string prefixed with "0x", for example:

```
>>> hex(255)
'0xff'
>>> hex(-42)
'-0x2a'
```

```
>>>
```

If `x` is not a Python `int` object, it has to define an `__index__()` method that returns an integer.

See also `int()` for converting a hexadecimal string to an integer using a base of 16.

진법 변환

- 10진수 -> 2, 8, 16진수 변환



decimal_to_binary.py

File Edit Format Run Options Window Help

```
x = 100000
```

```
print()
```

```
print("%d is %s in binary " %(x, bin(x)))
```

```
print("%d is %s in binary " %(x, oct(x)))
```

```
print("%d is %s in binary " %(x, hex(x)))
```

```
100000 is 0b11000011010100000 in binary
```

```
100000 is 0o303240 in binary
```

```
100000 is 0x186a0 in binary
```

```
>>>
```

진법 변환

- N진수 -> 10진수 변환($2 \leq n \leq 32$)

```
class int(x=0)
```

```
class int(x, base=10)
```

Return an integer object constructed from a number or string `x`, or return `0` if no arguments are given. If `x` is a number, return `x.__int__()`. For floating point numbers, this truncates towards zero.

If `x` is not a number or if `base` is given, then `x` must be a string, `bytes`, or `bytearray` instance representing an *integer literal* in radix `base`. Optionally, the literal can be preceded by `+` or `-` (with no space in between) and surrounded by whitespace. A base-`n` literal consists of the digits 0 to `n-1`, with `a` to `z` (or `A` to `Z`) having values 10 to 35. The default `base` is 10. The allowed values are 0 and 2–36. Base-2, -8, and -16 literals can be optionally prefixed with `0b/0B`, `0o/0O`, or `0x/0X`, as with integer literals in code. Base 0 means to interpret exactly as a code literal, so that the actual base is 2, 8, 10, or 16, and so that `int('010', 0)` is not legal, while `int('010')` is, as well as `int('010', 8)`.

The integer type is described in [Numeric Types — int, float, complex](#).

Changed in version 3.4: If `base` is not an instance of `int` and the `base` object has a `base.__index__` method, that method is called to obtain an integer for the base. Previous versions used `base.__int__` instead of `base.__index__`.

Changed in version 3.6: Grouping digits with underscores as in code literals is allowed.

진법 변환

- N진수 -> 10진수 변환($2 \leq n \leq 32$)

```
n_to_decimal.py
File Edit Format Run Options Window Help

xb = "11000011010100000"
print()
print("%s is %d in decimal" % (xb, int(xb,2)))

xo = "303240"
print("%s is %d in decimal" % (xo, int(xo,8)))

xh = "186a0"
print("%s is %d in decimal" % (xh, int(xh,16)))
```

```
11000011010100000 is 100000 in decimal
303240 is 100000 in decimal
186a0 is 100000 in decimal
>>>
```

심화학습

- bit
 - Computer의 기본 정보 단위
 - '0'과 '1' 표현
 - 전기가 들어오면 '1', 나가면 '0'
 - 3 bit로 표현할 수 있는 정보양
 - 000, 001, 010, 011, 100, 101, 110, 111 → 8개
 - 정보의 양 → 구분할 수 있는 아이템 개수
 - n bit → 2^n 표현 가능 ($0 \sim 2^n - 1$)
 - 8 bit = 1 byte
 - 0 ~ 255 표현 가능
 - 32 bit 운영 체제: 2^{32} 인식
 - 64 bit 운영 체제: 2^{64} 인식

심화학습

- 2의 지수승
 - 1 byte $\rightarrow 2^8$
 - 10 bit $\rightarrow 2^{10} = 1,024 \approx 1,000$ (Kilo)
 - 20 bit $\rightarrow 2^{20} = (2^{10})^2 = 1,048,576 \approx 1,000,000 = 1$ Mega
 - 30 bit $\rightarrow 2^{30} = (2^{10})^3 = 1,073,741,824 \approx 1,000,000,000 = 1$ Giga
 - 32 bit $\rightarrow 2^{32} = (2^{10})^3 \times 2^2 = 4,294,967,296 \approx 4,000,000,000 = 4$ Giga
 - ...
 - 64 bit $\rightarrow 2^{64} = (2^{10})^6 \times 2^4 = 18,446,744,073,709,551,616 \approx 18$ Exa
- 32 bit computer는 최대 4GB(Giga Byte)의 메모리만 장착할 수 있음
 - 정확히는 더 큰 메모리를 장착해도 4 GB만 인식하여 사용할 수 있다. 그 이유에 대해서 생각해 보시오.
- 64 bit CPU Max Memory Size

<https://www.compuram.de/blog/en/how-much-ram-can-be-addressed-under-the-current-32-bit-and-64-bit-operating-systems/>

심화학습

Maximum addressable memory

← → ↺

compuram.de/blog/en/how-much-ram-can-be-addressed-under-the-current-32-bit-and-64-bit-operating-systems/

🔍 ☆ 👤 ⋮

Operating System	Maximum Memory (RAM)
Mac OS X from 10.9	128GB
Mac OS X < 10.9	96GB
Windows 10 Home 32-Bit	4GB
Windows 10 Home 64-Bit	128GB
Windows 10 Pro 32-Bit	4GB
Windows 10 Pro 64-Bit	2TB
Windows 10 Pro for Workstation 32-Bit	4GB
Windows 10 Pro for Workstation 64-Bit	6TB
Windows 10 Enterprise 32-Bit	4GB
Windows 10 Enterprise 64-Bit	6TB
Windows 10 Education 32-Bit	4GB
Windows 10 Education 64-Bit	128GB
Windows 8.1 Pro 32-Bit	4GB
Windows 8.1 Enterprise 32-Bit	4GB
Windows 8.1 64-Bit	192GB
Windows 8.1 Pro 64-Bit	512GB
Windows 8.1 Enterprise 64-Bit	512GB
Windows RT 8 32-Bit	4GB
Windows 8 32-Bit	4GB
Windows 8 Pro 32-Bit	4GB
Windows 8 Enterprise 32-Bit	4GB
Windows 8 64-Bit	192GB
Windows 8 Pro 64-Bit	512GB
Windows 8 Enterprise 64-Bit	512GB
Windows 7 Starter 32-Bit	2GB

Extended product availability view in CompuRAM's web shop

NEWSLETTER

Subscribe

What's the benefit of a RAM/SSD upgrade?
Our performance tests show what you can squeeze out of your system ▶

SYSTEM BENCHMARK

00:23:50

Windows 10 Pro 64-Bit

8GB 16GB 32GB 64GB

2018-03-01 14:00:00

TAGS

Apple (7)

CompuRAM News (17)

Field tests (2)

Installation (10)

Limits (5)

Market Activity (2)

Notebook (11)

Operating Systems (4)

Popular Errors (6)

Product news (14)

Purchasing Advices (14)

RAM test (7)

RAMinator (5)

Server (10)

SSD (18)

Technology (9)

Troubleshooting (9)


Video (7)

↑

★★★★★

Excellent

4.92/5.00

 TRUSTED SHOPS

심화 학습

- 국제단위계 - SI접두어

10^n	접두어	기호	배수
10^{24}	요타 (yotta)	Y	자
10^{21}	제타 (zetta)	Z	십 <u>해</u>
10^{18}	엑사 (exa)	E	백 <u>경</u>
10^{15}	페타 (peta)	P	천조
10^{12}	테라 (tera)	T	<u>조</u>
10^9	기가 (giga)	G	<u>십억</u>
10^6	메가 (mega)	M	<u>백만</u>
10^3	킬로 (kilo)	K	<u>천</u>
10^2	헥토 (hecto)	H	<u>백</u>
10^1	데카 (deca)	Da	<u>십</u>
10^0			<u>일</u>

10^n	접두어	기호	배수
10^0			<u>일</u>
10^{-1}	데시 (deci)	d	십분의 일
10^{-2}	센티 (centi)	c	백분의 일
10^{-3}	밀리 (milli)	m	천분의 일
10^{-6}	마이크로 (micro)	μ	백만분의 일
10^{-9}	나노 (nano)	n	십억분의 일
10^{-12}	피코 (pico)	p	일조분의 일
10^{-15}	펨토 (femto)	f	천조분의 일
10^{-18}	아토 (atto)	a	백경분의 일
10^{-21}	zepto (zepto)	z	십해분의 일
10^{-24}	욕토 (yocto)	y	일자분의 일

실습 – homework

- 오차율 계산
 - 10 bit로 표현하는 kilo($2^{10} = 1,024$)와 실제 1,000과의 오차율을 계산해라.
 - $(2^{10}-1000) / 1000$
 - 20 bit에 대해서도 오차율을 계산하라.
 - 30 bit에 대해서도 오차율을 계산하라.

Homework

- 다음 수업 시작 전까지 ecampus로 제출
- 제출방법

파일명 : 과소사-학번-이름-chapter번호

예) 과소사-20201234-김국민-2.1장

- 파일이 여러 개일 경우 zip으로 묶어서 제출