```
#include <cmath>
```

std::floor()

std::ceil()

std::abs()

std::cos()

std::exp()

std::log()

std::pow()

std::sqrt()

std::round

std::sin()

std::tan()

Reference Doc

https://en.cppreference.com/w/cpp/header/cmath

```
double weight { 7.7 };

//floor
std::cout << "Weight rounded to floor is : " << std::floor(weight) << std::endl;

//ceil
std::cout << "Weight rounded to ceil is : " << std::ceil(weight) << std::endl;
```

```
//abs
double savings {-5000 };
weight = 7.7;
std::cout << "Abs of weight is : " << std::abs(weight) << std::endl;
std::cout << "Abs of savings is : " << std::abs(savings) << std::endl;
```

$$\exp(\mathrm{x}) = \mathrm{e}^{x}$$

```
//exp : f(x) = e ^ x , where e = 2.71828 .
double exponential = std::exp(10);
std::cout << "The exponential of 10 is : " << exponential << std::endl;
```

```
//pow
std::cout << "3 ^ 4 is : " << std::pow(3,4) << std::endl;
std::cout << "9^3 is : " << std::pow(9,3) << std::endl;
```

```
//log : reverse function of pow. if 2^3 = 8 , log 8 in base 2 = 3.  Log is like asking
// to which exponent should we elevate 2 to get eight ? Log, by default computes the log
// in base e. There also is another function which uses base 10 called log10

// Try the reverse operation of  e^4 = 54.59 , it will be log 54.59 in base e = ?
std::cout << "Log ; to get 54.59, you would elevate e to the power of : " << std::log(54.59) << std::endl;

//log10 , 10 ^ 4 = 10000  , to get 10k , you'd need to elevate 10 to the power of ? , this is log in base 10
std::cout << "To get 1000, you'd need to elevate 10 to the power of : " << std::log10(10000) << std::endl;
```

```
//sqrt
std::cout << "The square root of 81 is : " << std::sqrt(81) << std::endl;

//round. Halfway points are rounded away from 0. 2,5 is rounded to 5 for example
std::cout << "3.654 rounded to : " << std::round(3.654) << std::endl;
std::cout << "2.5 is rounded to : " << std::round(2.5) << std::endl;
std::cout << "2.4 is rounded to : " << std::round(2.4) << std::endl;
```

```
//sqrt
std::cout << "The square root of 81 is : " << std::sqrt(81) << std::endl;

//round. Halfway points are rounded away from 0. 2,5 is rounded to 5 for example
std::cout << "3.654 rounded to : " << std::round(3.654) << std::endl;
std::cout << "2.5 is rounded to : " << std::round(2.5) << std::endl;
std::cout << "2.4 is rounded to : " << std::round(2.4) << std::endl;
```

Trigonometric functions	
`sin` `sinf` (C++11) `sinl` (C++11)	computes sine ($\sin x$) (function)
`cos` `cosf` (C++11) `cosl` (C++11)	computes cosine ($\cos x$) (function)
`tan` `tanf` (C++11) `tanl` (C++11)	computes tangent ($\tan x$) (function)
`asin` `asinf` (C++11) `asinl` (C++11)	computes arc sine ($\arcsin x$) (function)
`acos` `acosf` (C++11) `acosl` (C++11)	computes arc cosine ($\arccos x$) (function)
`atan` `atanf` (C++11) `atanl` (C++11)	computes arc tangent ($\arctan x$) (function)
`atan2` `atan2f` (C++11) `atan2l` (C++11)	arc tangent, using signs to determine quadrants (function)

Standard library header <cmath>

This header was originally in the C standard library as **<math.h>**.

This header is part of the numeric library.

Types

float_t (C++11)	most efficient floating-point type at least as wide as `float` (typedef)
double_t (C++11)	most efficient floating-point type at least as wide as `double` (typedef)

Macros

HUGE_VALF (C++11) **HUGE_VAL** **HUGE_VALL** (C++11)	indicates the overflow value for `float`, `double` and `long double` respectively (macro constant)
INFINITY (C++11)	evaluates to positive infinity or the value guaranteed to overflow a `float` (macro constant)
NAN (C++11)	evaluates to a quiet NaN of type `float` (macro constant)
math_errhandling (C++11) **MATH_ERRNO** (C++11) **MATH_ERREXCEPT** (C++11)	defines the error handling mechanism used by the common mathematical functions (macro constant)

Classification