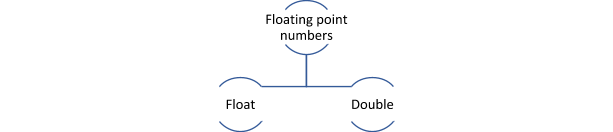
float and double are primitive data types used by [programming languages](https://hackr.io/blog/what-is-programming-language) to store floating-point real (decimal) numbers like 10.923455, 433.45554598 and so on. This article will give you the detailed difference between float and double data type.

## Float vs Double: Difference You should know

In the computing world, numeric data can be represented in 2 ways – fixed-point and floating-point arithmetic. Fixed point data is what we call as the integer with some sort of scaling factor. For example, 3.14 will be 314 with a scaling factor of 100, or 2.3456 can be rounded to 2.345 to achieve a fixed number of digits. This method compromises on the accuracy of the result and is not suitable in all situations.

It is thus easier and more accurate to use floating-point representation for high-precision calculations (as we will see in the next section).



If you want to read about floating-point numbers from an arithmetic point of view, you can read this [Wikipedia article](https://en.wikipedia.org/wiki/Floating-point_arithmetic" \t "https://hackr.io/blog/_blank). For this blog, we have focussed on how programming languages use these floating-point numbers to get precision values and range.

### **Why use floating point data?**

#### Mixed precision

In mixed-precision, single-precision values can be used for complex calculations. You can consider it as a trade-off between accuracy and memory efficiency. By combining float16 (half precision) and float32 (single-precision), applications can increase application performance and data transfer speed. Half precision was defined by Microsoft and Nvidia to save space and complexity of floating-point calculations.

But, not always.

#### Arbitrary precision

We use floating-point and integer math for precise calculations where the result is only limited by the amount of memory available on the system. This type of calculation is called as arbitrary or infinite precision calculation. One of the most important applications of arbitrary precision is public-key cryptography where computations with numbers having hundreds of digits (exponential) are a common sight.

Another similar precision type is the [symbolic computation](https://en.wikipedia.org/wiki/Computer_algebra" \t "https://hackr.io/blog/_blank) where we use exact values of symbols (like PI) for complex computations.

* Single Precision is a 32-bit type; Double Precision is a 64-bit type. Accordingly, Doubles store a much broader range of values, and with much more precision. The floating point format is comprised of a sign bit, exponent bits (which give you range) and fraction bits (which give you numerical precision).

# Difference between Single Precision and Double Precision

* **Difficulty Level :** [Easy](https://www.geeksforgeeks.org/easy/)
* **Last Updated :** 09 May, 2020

According to IEEE standard, [floating-point number is represented in two ways](https://www.geeksforgeeks.org/floating-point-representation-basics/):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Precision** | **Base** | **Sign** | **Exponent** | **Significand** |
| Single precision | 2 | 1 | 8 | 23+1 |
| Double precision | 2 | 1 | 11 | 52+1 |

**1. Single Precision:**  
Single Precision is a format proposed by IEEE for representation of floating-point number. It occupies 32 bits in computer memory.

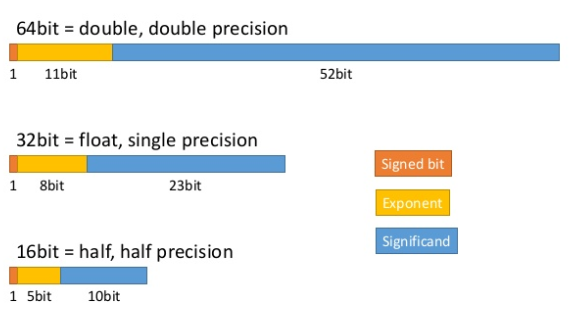


**2. Double Precision:**  
Double Precision is also a format given by IEEE for representation of floating-point number. It occupies 64 bits in computer memory.



**Difference between Single and Double Precision:**

| SINGLE PRECISION | DOUBLE PRECISION |
| --- | --- |
| In single precision, 32 bits are used to represent floating-point number. | In double precision, 64 bits are used to represent floating-point number. |
| It uses 8 bits for exponent. | It uses 11 bits for exponent. |
| In single precision, 23 bits are used for mantissa. | In double precision, 52 bits are used for mantissa. |
| Bias number is 127. | Bias number is 1023. |
| Range of numbers in single precision :**2^(-126)** to **2^(+127)** | Range of numbers in double precision : **2^(-1022)** to **2^(+1023)** |
| This is used where precision matters less. | This is used where precision matters more. |
| It is used for wide representation. | It is used for minimization of approximation. |
| It is used in simple programs like games. | It is used in complex programs like scientific calculator. |
| This is called binary32. | This is called binary64. |



### **Float and double**

Double is more precise than float and can store 64 bits, double of the number of bits float can store.

Double is more precise and for storing large numbers, we prefer double over float. For example, to store the annual salary of the CEO of a company, double will be a more accurate choice. All trigonometric functions like sin, cos, tan, mathematical functions like sqrt return double values. However, double comes with a cost. Unless we do need precision up to 15 or 16 decimal points, we can stick to float in most applications, as double is more expensive. It takes about 8 bytes to store a variable. We append ‘f’ or ‘F’ to the number to indicate that it is float type failing which it is taken as double.

A small table that gives the memory requirement and range of float and double is shown below –

|  |  |  |
| --- | --- | --- |
| Floating point type | Memory requirement | Range |
| Float | 4 bytes | ±3.40282347E+38F i.e. 6-7 significant digits |
| Double | 8 bytes | ±1.79769313486231570E+308 i.e. 15-16 significant digits |

Float and double function in the same way in all programming languages. For example, in Java, both will throw NumberFormatException with operations where actual numbers are not involved. Note that the compiler will not detect this exception.

String sname = "DR";float fname = Float.parseFloat(sname);

System.out.println(fname/num1); // Leads to NumberFormatException at runtime

Dividing float and double by zero will give an output of ‘Infinity’ in Java.double num2 = 344.55555555;

System.out.println(num2/0);

This won’t result in an error but is an invalid operation (NaN). Learn more about NaN [here](https://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html" \l "jls-4.2.3" \t "https://hackr.io/blog/_blank).

Where will we use precision values?

Almost everywhere!

If you work with small quantities of data – like average marks, area of triangle etc… use double by default. But, if you deal with a lot of numbers where high precision is involved and any rounding off can change results – like trigonometry, width of a human hair, neural networks, spin of an electron, coordinates of a location and so on – it is important to know about the differences between float and double. While Java encourages you to use double, in languages like C you have the flexibility of using whichever you want.

A typical java declaration will look like –

float number1 = (float) 12.211111111111;

Now, if you do not do that typecasting, your code will not compile in Java. When you do it, while printing the number, you will only get 6 digits after the decimal point.

Consider a simple program of multiplying two numbers – 1.42222\*234.56433 written in C.

This could be anything like atomic mass or gravitational force which has to have all its significant digits intact!

float num1 = 1.42222\*234.56433;double num2 = 1.42222\*234.56433;printf("%f", num1);printf("%.10f", num2);

While num1 returns the value as 333.602081, num2 declared as double returns 333.6020814126, which is precise upto 10 digits as mentioned in our printf statement. We can also print a float as a double and vice versa, it all depends on how we write the printf statement. Writing %f will strip off some significant digits, while when we specify number of digits, the entire value up till that will be printed. To print the value in exponential terms, you should use “%e”.

In Java, as we have seen earlier, only if we typecast to (float), the value is printed. Java takes all high precision decimal values as double by default.

float values;double doubes;

values = (float) (1.42222\*234.56433);

doubes = 1.42222\*234.56433;

System.out.println(values);

System.out.println(doubes);

will yield 333.60208 and 333.6020814126 respectively.

**Logical comparisons**

We use the operators <, <=, >= and > to compare float and double values. With integers, we can use != and = but here we don’t because the latter operators are less precise.

Obviously, when float is used, exact comparison is not possible as precision is only till 5-6 digits. Any differences in the numbers is not caught.

float number1 = (float) 3.1434343;float number2 = (float) 3.1434343333;

if(number1 == number2)

System.out.println("equal");

else

System.out.println("not equal");

double number3 = 3.1434343;double number4 = 3.1434343333;

if(number3 == number4)

System.out.println("equal");

else

System.out.println("not equal");

What do you think the output will be?

You might have guessed it – the first one will give “equal”, while the second one will give “not equal”.

To avoid the typecasting every time we write the number in float, we can suffix the number with ‘f’. For example,

float number1 = 3.1434343**f**;

### **Big Decimal**

.NET and Java also have Decimal/BigDecimal class that has higher precision than double. For more accurate calculations like in financial and banking applications, Decimal is used because it further reduces rounding errors.

### **long double**

Some programming languages like C use long double that gives more precision than double. Check out the different [data types of C](https://hackr.io/blog/c-data-types).

### **Division with float and double**

Same as in multiplication or addition, the division will give more precision digits in double. Consider this simple example –

float number1 = 3.1434343f;double number2 = 3.1434343;float divide = 22/7f;// first let us print the result as doubledouble result1 = number1/divide;/\* the same result but now it is a float value,

note the difference in significant digits \*/float result3 = number1/divide;// the double valuedouble result2 = number2/divide;

System.out.println(result1); // 1.0001837015151978

System.out.println(result3); // 1.0001837

System.out.println(result2); // 1.000183662587488

This is particularly useful when denominator is bigger than numerator and the result is in small fractions like –

float pie = 22/7f;float pieby4096 = pie/4096;double dpie = 22/7d;double dpieby4096 = dpie/4096;

System.out.println("Float Pie is - " + pie);

System.out.println("Double pie is - " + dpie);

System.out.println("Float Pie divided by 4096 - " + pieby4096);

System.out.println("Double Pie divided by 4096 - " + dpieby4096);double pieby4096usingfloatpie = pie/4096;

System.out.println("Float Pie divided by 4096 with result as double - " + pieby4096usingfloatpie);

See the results –

Float Pie is - 3.142857

Double pie is - 3.142857142857143

Float Pie divided by 4096 - 7.672991E-4

Double Pie divided by 4096 - 7.672991071428571E-4

Float Pie divided by 4096 with result as double - 7.672990905120969E-4

Pay attention to the last 3 results. The pie that we choose to divide the numbers makes a difference in the significant digits of the result. This is exactly the precision we are talking about!

### **Concatenation with String**

In Java, it is possible to concatenate strings with double and float using + operator.

String str = "test";float flo = 23.2f;

String concat = str + flo;double dou = 3.45555555;

concat += dou;

System.out.println(concat); // result will be test23.23.45555555

### **Float vs Double: Head to head comparison**

Now that we know what float and double are, it will be good to create a table of differences for quick reference and recap.

|  |  |
| --- | --- |
| **Float** | **Double** |
| Single precision value | Double precision value |
| Can store Up to 7 significant digits | Stores up to 15 significant digits |
| Occupies 4 bytes of memory (32 bits IEEE 754) | Occupies 8 bytes of memory (64-bits IEEE 754) |
| If more than 7 digits are present, value is rounded off | 7-15 digits are stored as they are |
| With Java, one needs to typecast to declare float –  float fnum = 2.344f;  or  float fnum = (float) 2.344; | Double is the default decimal point type for Java.  double dnum = 2.344; |
| If high precision is not required and the program only needs a huge array of decimal numbers to be stored, float is a cost-effective way of storing data and saves memory. | Double is costlier, occupies more space and is more effective when more precision is required. For example, currency conversion, financial reports and transactions, scientific calculations etc… |

### 2

Single precision is the 32 bit representation of numerical values in computers. It is also known as binary32. Some languages, like JAVA, C++ use float to store these kinds of numerals. Some languages (Visual Basic) refer single numerals as single.

Single precision is most widely used because of its capability to represent wide range of numeral values, though it reduces the amount of precision achieved.

Single precision uses 32 bit to represent a floating point number.

First bit represent the sign of the number, negative or positive.

Next 8 bits are used to store the exponent of the number. This exponent can be signed 8-bit integer ranging from -127 – 128 of signed integer (0 to 255).

And the left 23 bits are used to represent the fraction part and are called fraction bits.

8 exponent bits provide us with the range and 23 bits provide us with the actual precision.

### Double Precision

Double precision is called binary64. Double precision uses 64 bits to represent a value.

First bit is used for the same purpose as in single point precision i.e., represents sign of the number.

Next 11 bits are used to denote exponent, which provide us with the range, and has 3 more bits than single precision, so it is used to represent a wider range of values.

Next 52 bits are used to represent the fractional part which is 29 bits more than binary32 bit representation scheme. So it has a greater precision than single precision.

Double floating point precision are used where high arithmetic precision is required and number like – 2/19 have to be used. Double point precision requires more memory as compared to single precision, hence are not useful when normal calculations are to be performed. This representation technique finds its use in the scientific calculations.

### short

So far we have looked at qualifiers that increase the storage space, and thereby the value range, of data types. The short qualifier can be used to reduce the storage space and range of the int data type. This effectively reduces the integer to 16-bits in width, limiting the signed value range to −32,768 to +32,767:

short int myshort;

In fact, signed or unsigned are "types" attributed to the exact same bit patters, and how these bit patterns are interpreted when compared or promoted is defined by their respective types. For example: unsigned char u = 0xFF; // decimal 255 signed char s = 0xFF; // decimal -1. You can see how both values are the sam