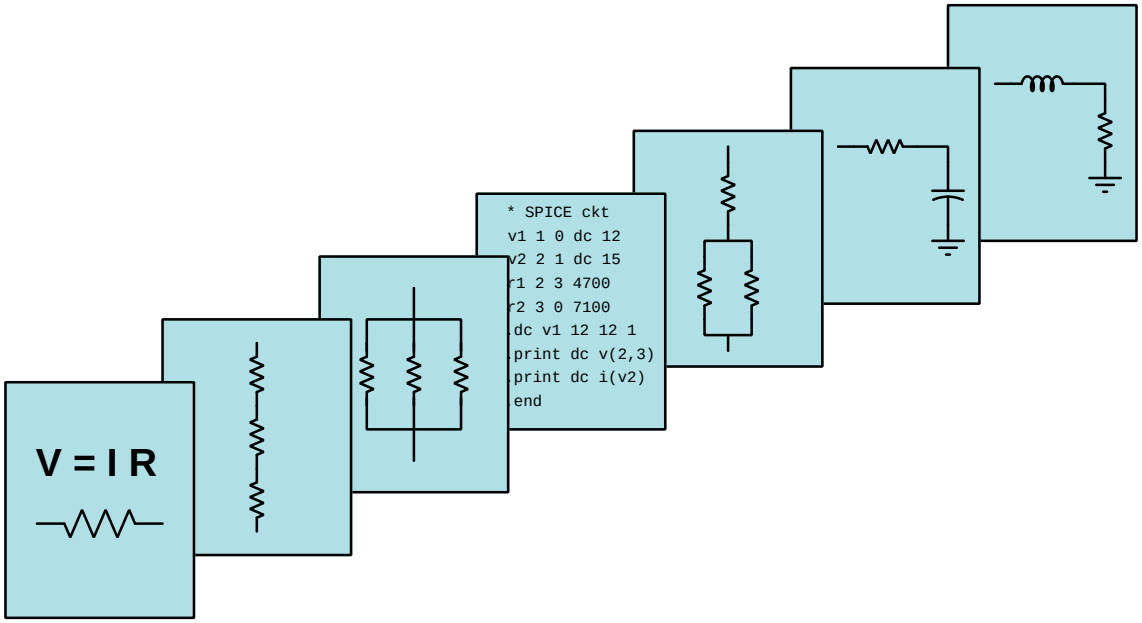
Modular Electronics Learning (ModEL) project



Basic Trade Mathematics

© 2021-2024 by Tony R. Kuphaldt – under the terms and conditions of the Creative Commons Attribution 4.0 International Public License

Last update = 3 February 2024

This is a copyrighted work, but licensed under the Creative Commons Attribution 4.0 International Public License. A copy of this license is found in the last Appendix of this document. Alternatively, you may visit `http://creativecommons.org/licenses/by/4.0/` or send a letter to Creative Commons: 171 Second Street, Suite 300, San Francisco, California, 94105, USA. The terms and conditions of this license allow for free copying, distribution, and/or modification of all licensed works by the general public.

Contents

Chapter 1

Introduction

Chapter 2

Tutorial

2.1 Types of numbers

- **Natural numbers** – the numbers children first learn for counting (e.g. 1, 2, 3, 4 $\cdots$)
- **Whole numbers** – all natural numbers as well as zero (e.g. 0, 1, 2, 3, 4 $\cdots$)
- **Integers** – all whole numbers and their negative counterparts (e.g. $\cdots$, -3, -2, -1, 0, 1, 2, 3, $\cdots$)
- **Rational numbers** – any quantity that may be expressed as a quotient (division) of integers, which include all integers (e.g. $\cdots$, $-\frac{3}{7}$, $-\frac{1}{200}$, $\frac{0}{5}$, $\frac{3}{2}$, $\frac{14}{7}$, $\cdots$)
- **Irrational numbers** – a quantity that cannot be expressed as a quotient (division) of integers (e.g. $\sqrt{2}$, π, etc.). When written in decimal form, these are the numbers whose trailing digits go on forever without repeating any pattern.
- **Real numbers** – the collection of all rational and irrational numbers.
- **Imaginary numbers** – any quantity involving the square root of a negative number (e.g. $\sqrt{-1}$).
- **Complex numbers** – the collection of real and imaginary numbers.

2.2 Basic operations

The four basic *operations* of arithmetic are:

- Addition
- Subtraction
- Multiplication
- Division

Addition is simply the direct accumulation of values, for example: $3 + 5 = 8$

Subtraction is the opposite of addition, where each subtracting quantity takes away from the total. For example: $14 - 5 = 9$. Subtraction may also be viewed as the addition of a negative number. For example, we could alternatively express $14 - 5 = 9$ as $14 + (-5) = 9$

Multiplication is repeated addition. For example, $4 \times 5 = 4 + 4 + 4 + 4 + 4 = 20$. An alternative representation for multiplication is to place the two quantities adjacent to each other and separated by parentheses, for example: $(4)(5) = 20$. Less common is to separate the two quantities by a center-dot symbol, for example: $4 \cdot 5 = 20$. In computer programming the standard symbol for multiplication is the star (`*`) character.

Division is the opposite of multiplication, where the dividing quantity splits the total into that many pieces and the result is the size of each of those pieces. For example, $36 \div 12 = 3$. An alternative representation for division is to place the two quantities on top and bottom of a horizontal line segment, for example: $\frac{36}{12} = 3$. In computer programming the standard symbol for division is the forward-slash (`/`) character.

You should notice already a sort of symmetry between some of these operations, in that one always works to "un-do" another. For example, subtraction un-does addition: if $3 + 5 = 8$ then $8 - 5 = 3$. Likewise, division un-does multiplication: if $4 \times 5 = 20$ then $20 \div 5 = 4$. These are called *complementary operations*, and this is a central concept in the application of *algebra*.

Reciprocation is a form of division, representing the quotient formed by one divided by the given number, for example the reciprocal of 4 is $\frac{1}{4} = 0.25$. Reciprocation is its own complementary function, as reciprocating a reciprocated number yields that original number. For example: if $\frac{1}{4} = 0.25$ then $\frac{1}{0.25} = 4$

From these four elementary arithmetic operations we may build other operations as well:

- Powers
- Roots
- Exponents
- Logarithms

A power is repeated multiplication. For example, $3^5 = 3 \times 3 \times 3 \times 3 \times 3 = 243$. Negative powers are valid too, and simply represent the reciprocal of the positive power. For example: $2^{-3} = \frac{1}{2^3} = \frac{1}{8} = 0.125$. Certain powers have common names, for example anything raised to the second power is a *square*, while anything raised to the third power is a *cube*. For example the square of three is nine ($3^2 = 9$) and the cube of two is eight ($2^3 = 8$).

If we divide one power by another power, the result is a power that is the difference of the degrees. For example: $\frac{4^5}{4^3} = \frac{4 \times 4 \times 4 \times 4 \times 4}{4 \times 4 \times 4} = 4^{5-3} = 4 \times 4 = 4^2 = 16$. This logically leads to the non-intuitive conclusion that any quantity raised to the power of zero is one, because a power of zero simply means the quotient of two identical powers. For example: $\frac{4^3}{4^3} = 4^{3-3} = 4^0 = 1$

A root is the opposite (complementary operation) of a power. Taking the root of a quantity yields the value you would have to raise by that specified power to get the original quantity. An unadorned "radicand" symbol implies the second root, or *square root*. For example, if $4^2 = 16$ then $\sqrt{16} = 4$; if $2^3 = 8$ then $\sqrt[3]{8} = 2$. For even-valued roots such as square roots, the result actually takes the form of both a positive and a negative number. For example: $\sqrt{16}$ has two answers, 4 and -4.

An exponent is also repeated multiplication, and indeed is the same basic concept as a power. The major difference between powers and exponents becomes evident when we change one of the values but keep the other constant. For example, if we vary the "base" number while keeping the superscript constant (e.g. 2^2, 3^2, 4^2, 5^2) we refer to it as a power function, but if we keep the "base" number constant while varying the superscript (e.g. 2^2, 2^3, 2^4, 2^5) we refer to it as an exponential function.

A logarithm is the opposite (complementary operation) of an exponent. Taking the logarithm of a quantity yields the exponent necessary to raise the specified base value to get that original quantity. Two standardized types of logarithms exist: the *common logarithm* (log) where 10 is the assumed base, and the *natural* logarithm (ln) where e is the assumed base. For example: if $10^3 = 1000$ then $\log 1000 = 3$. Similarly, $\ln(e^4) = 4$.

2.3 Names of operational objects

Addition

Addend *Addend* *Sum*
$3 + 5 = 8$

Subtraction

Minuend *Subtrahend* *Difference*
$3 - 5 = -2$

Multiplication

Multiplicand *Multiplier* *Product*
$3 \times 5 = 15$

Division

Dividend *Divisor* *Quotient*
$12 \div 3 = 4$

Exponents/Powers

Exponent *Argument*
$10^{3} = 1000$
Base

Logarithms

Argument *Exponent*
$\log_{10} 1000 = 3$
Base

Roots

Index *Root*
$\sqrt[3]{8} = 2$
Radicand

2.4 Properties and identities of real numbers

In the following examples, every letter represents a real number, with repeated letters representing the same quantities.

Additive identity:

$$a + 0 = a$$

Additive inverse:

$$a + (-a) = 0$$

Multiplicative identity:

$$a \times 1 = a$$

Multiplicative inverse:

$$a \times \frac{1}{a} = 1$$

Transitive property

$$\text{if } a = b \text{ and } b = c \text{ then } a = c$$

Associative property of addition:

$$a + (b + c) = (a + b) + c$$

Associative property of multiplication:

$$a(bc) = (ab)c$$

Commutative property of addition:

$$a + b = b + a$$

Commutative property of multiplication:

$$ab = ba$$

Distributive property:

$$a(b + c) = ab + bc$$

Properties of exponents:

$$a^x a^y = a^{x+y}$$

$$\frac{a^x}{a^y} = a^{x-y}$$

$$(ab)^x = a^x b^x$$

$$\left(\frac{a}{b}\right)^x = \frac{a^x}{b^x}$$

$$(a^x)^y = a^{xy}$$

Properties of roots:

$$(\sqrt[x]{a})^x = a$$

$$\sqrt[x]{a^x} = a \text{ if } a \geq 0$$

$$\sqrt[x]{ab} = \sqrt[x]{a}\sqrt[x]{b}$$

$$\sqrt[x]{\frac{a}{b}} = \frac{\sqrt[x]{a}}{\sqrt[x]{b}}$$

2.5 Fractions

A fraction represents a specified portion of some base quantity. For example, the fraction $\frac{3}{5}$ means a proportion equal to three parts out of five. A fraction comprised of equal values in the numerator (upper) and denominator (lower) places (e.g. $\frac{7}{7}$ or $\frac{-12}{-12}$) is equal to one. A fraction may also be thought of as an un-completed division problem, e.g. $\frac{3}{5}$ means "three divided by five" which happens to be equal to 0.6.

Fractions may be directly added to and subtracted from each other if they all share the same denominator. So long as there is a common denominator, the numerators simply add and subtract like regular numbers. For example:

$$\frac{4}{9} + \frac{3}{9} = \frac{4+3}{9} = \frac{7}{9}$$

Or,

$$\frac{10}{14} + \frac{1}{14} - \frac{3}{14} = \frac{10+1-3}{14} = \frac{8}{14}$$

Fractions may be directly multiplied regardless of their denominators. For example:

$$\frac{2}{3} \times \frac{4}{5} = \frac{2 \times 4}{3 \times 5} = \frac{8}{15}$$

If we wish to add fractions that do not share a common denominator, we may multiply one or more of those fractions by a "unity" fraction (i.e. a fraction with equal numerator and denominator having an over-all value of one) to make the denominators equal without changing the actual value of the multiplied fraction(s), since any real number multiplied by one is the original value (i.e. the multiplicative identity). For example,

$$\frac{2}{3} + \frac{1}{6} = \left(\frac{2}{3} \times \frac{2}{2}\right) + \frac{1}{6} = \frac{4}{6} + \frac{1}{6} = \frac{5}{6}$$

Reciprocation of a fraction means turning it upside-down so that what was the numerator is now the denominator and vice-versa. For example, the reciprocal of $\frac{5}{2}$ is $\frac{2}{5}$.

Division by a fraction is the same thing as multiplication by its reciprocal. For example:

$$\frac{6}{7} \div \frac{1}{3} = \frac{6}{7} \times \frac{3}{1} = \frac{6 \times 3}{7 \times 1} = \frac{18}{7}$$

If a fraction comprised of products contains identical multipliers in both the numerator and denominator, those will cancel out. For example, the fraction $\frac{3 \times 7}{2 \times 3}$ simplifies to $\frac{7}{2}$ because the "3" multiplier present on both top and bottom cancel out. The reason why may be shown as follows:

$$\frac{3 \times 7}{2 \times 3} = \frac{7 \times 3}{2 \times 3} = \frac{7}{2} \times \frac{3}{3} = \frac{7}{2} \times 1 = \frac{7}{2}$$

A *proper fraction* is one where the numerator is less than the denominator. Examples include $\frac{3}{8}$, $\frac{-5}{19}$, and $\frac{6}{-11}$. An *improper fraction* is one where the numerator exceeds the denominator (e.g. $\frac{11}{4}$, $\frac{-17}{12}$, and $\frac{13}{-3}$). A *mixed fraction* is a combination of a fraction and a whole number, for example $3\frac{5}{6}$.

Translating from an improper fraction to a mixed fraction merely involves properties of fraction we have already seen. For example, we see here how the improper fraction $\frac{11}{4}$ converts to the mixed fraction $2\frac{3}{4}$:

$$\frac{11}{4} = \frac{4+4+3}{4} = \frac{4}{4} + \frac{4}{4} + \frac{3}{4} = 1 + 1 + \frac{3}{4} = 2\frac{3}{4}$$

Likewise, we may convert a mixed fraction into an equivalent (equal) improper fraction by using the same principles shown above, just in reverse. For example, converting $3\frac{5}{6}$ into its improper form:

$$3\frac{5}{6} = 1 + 1 + 1 + \frac{5}{6} = \frac{6}{6} + \frac{6}{6} + \frac{6}{6} + \frac{5}{6} = \frac{23}{6}$$

A fraction containing one or more other fractions within its numerator or denominator is called a *complex fraction*. These may be simplified quite easily if we just realize that the largest fraction bar may be treated as a division symbol. For example:

$$\frac{\frac{2}{9}}{\frac{5}{6}} = \frac{2}{9} \div \frac{5}{6} = \frac{2}{9} \times \frac{6}{5} = \frac{12}{45}$$

2.6 Unit conversions

Converting physical measurements from one unit to another is a common task for technical professionals, and it is a task that may be made simpler by using fractions. This technique involves setting up the original quantity as a fraction, then multiplying by a series of fractions having *physical* values of unity (1) so that by multiplication the original value does not change, but the units do. Let's take for example the conversion of quarts into gallons, an example of a fluid volume conversion:

$$35 \text{ qt} = ??? \text{ gal}$$

Now, most people know there are four quarts in one gallon, and so it is tempting to simply divide the number 35 by four to arrive at the proper number of gallons. However, the purpose of this example is to show you how the technique of unity fractions works, not to get an answer to a problem.

To demonstrate the unity fraction technique, we will first write the original quantity as a fraction, in this case a fraction with 1 as the denominator:

$$\frac{35 \text{ qt}}{1}$$

Next, we will multiply this fraction by another fraction having a *physical* value of unity (1) so that we do not alter[1] the quantity. This means a fraction comprised of equal measures in the numerator and denominator, but having different units of measurement. This "unity" fraction must be arranged in such a way that the undesired unit cancels out and leaves only the desired unit(s) in the product. In this particular example, we wish to cancel out quarts and end up with gallons, so we must arrange a fraction consisting of quarts and gallons having equal quantities in numerator and denominator, such that quarts will cancel and gallons will remain:

$$\left(\frac{35 \text{ qt}}{1}\right)\left(\frac{1 \text{ gal}}{4 \text{ qt}}\right)$$

Now we see how the unit of "quarts" cancels from the numerator of the first fraction and the denominator of the second ("unity") fraction, leaving only the unit of "gallons" left standing:

$$\left(\frac{35 \cancel{\text{qt}}}{1}\right)\left(\frac{1 \text{ gal}}{4 \cancel{\text{qt}}}\right) = 8.75 \text{ gal}$$

The reason this conversion technique is so powerful is it allows one to perform the largest range of unit conversions while memorizing the smallest possible set of conversion factors.

Here is a set of six equal volumes, each one expressed in a different unit of measurement:

1 gallon (gal) = 231.0 cubic inches (in^3) = 4 quarts (qt) = 8 pints (pt) = 128 fluid ounces (fl. oz.) = 3.7854 liters (l)

[1] A basic mathematical identity is that multiplication of any quantity by 1 does not change the value of that original quantity. If we multiply some quantity by a fraction having a physical value of 1, no matter how strange-looking that fraction may appear, the value of the original quantity will be left intact. The goal here is to judiciously choose a fraction with a physical value of 1 but with its units of measurement so arranged that we cancel out the original quantity's unit(s) and replace them with the units we desire.

Since all six of these quantities are physically equal, it is possible to build a "unity fraction" out of any two, to use in converting any of the represented volume units into any of the other represented volume units. Shown here are a few different volume unit conversion problems, using unity fractions built only from these factors (all canceled units shown using strike-out lines):

40 gallons converted into fluid ounces (using *128 fl. oz. = 1 gal* in the unity fraction):

$$\left(\frac{40 \text{ } \cancel{\text{gal}}}{1}\right)\left(\frac{128 \text{ fl. oz.}}{1 \text{ } \cancel{\text{gal}}}\right) = 5120 \text{ fl. oz}$$

5.5 pints converted into cubic inches (using *231 in^3 = 8 pt* in the unity fraction):

$$\left(\frac{5.5 \text{ } \cancel{\text{pt}}}{1}\right)\left(\frac{231 \text{ in}^3}{8 \text{ } \cancel{\text{pt}}}\right) = 158.8 \text{ in}^3$$

1170 liters converted into quarts:

$$\left(\frac{1170 \text{ } \cancel{\text{l}}}{1}\right)\left(\frac{4 \text{ qt}}{3.7854 \text{ } \cancel{\text{l}}}\right) = 1236 \text{ qt}$$

By contrast, if we were to try to memorize a 6×6 table giving conversion factors between *any two* of six volume units, we would have to commit 30 different conversion factors to memory! Clearly, the ability to set up "unity fractions" is a much more memory-efficient and practical approach.

This economy of conversion factors is very useful, and may also be extended to cases where linear units are raised to powers to represent two- or three-dimensional quantities. To illustrate, suppose we wished to convert 5.5 pints into cubic *feet* instead of cubic *inches*: with no conversion equivalence between pints and cubic feet included in our string of six equalities, what do we do?

We should know the equality between inches and feet: there are exactly 12 inches in 1 foot. This simple fact may be applied by incorporating *another* unity fraction in the original problem to convert cubic inches into cubic feet. We will begin by including another unity fraction comprised of 12 inches and 1 foot,just to see how this might work:

5.5 pints converted into cubic feet (*our first attempt!*):

$$\left(\frac{5.5 \text{ } \cancel{\text{pt}}}{1}\right)\left(\frac{231 \text{ in}^3}{8 \text{ } \cancel{\text{pt}}}\right)\left(\frac{1 \text{ ft}}{12 \text{ } \cancel{\text{in}}}\right) = 13.23 \text{ in}^2 \cdot \text{ft}$$

Unfortunately, this yields a non-sensical unit of square inch-feet. Even though $\frac{1 \text{ ft}}{12 \text{ in}}$ is a valid unity fraction, it does not *completely* cancel out the unit of cubic inches in the numerator of the first unity fraction. Instead, the unit of "inches" in the denominator of the unity fraction merely cancels out one of the "inches" in the "cubic inches" of the previous fraction's numerator, leaving square inches (in^2). What we need for full cancellation of cubic inches is a unity fraction relating *cubic* feet to *cubic* inches. We can get this, though, simply by *cubing* the $\frac{1 \text{ ft}}{12 \text{ in}}$ unity fraction:

5.5 pints converted into cubic feet (*our second attempt!*):

$$\left(\frac{5.5 \text{ pt}}{1}\right)\left(\frac{231 \text{ in}^3}{8 \text{ pt}}\right)\left(\frac{1 \text{ ft}}{12 \text{ in}}\right)^3$$

Distributing the third power to the interior terms of the last unity fraction:

$$\left(\frac{5.5 \text{ pt}}{1}\right)\left(\frac{231 \text{ in}^3}{8 \text{ pt}}\right)\left(\frac{1^3 \text{ ft}^3}{12^3 \text{ in}^3}\right)$$

Calculating the values of 1^3 and 12^3 inside the last unity fraction, then canceling units and solving:

$$\left(\frac{5.5 \text{ } \cancel{\text{pt}}}{1}\right)\left(\frac{231 \text{ } \cancel{\text{in}}^3}{8 \text{ } \cancel{\text{pt}}}\right)\left(\frac{1 \text{ ft}^3}{1728 \text{ } \cancel{\text{in}}^3}\right) = 0.0919 \text{ ft}^3$$

Now the answer makes sense: a volume expressed in units of cubic feet.

Once again, this unit conversion technique shows its power by minimizing the number of conversion factors we must memorize. We need not memorize how many cubic inches are in a cubic foot, or how many square inches are in a square foot, if we know how many linear inches are in a linear foot and we simply let the fractions "tell" us whether a power is needed for unit cancellation.

Unity fractions are also useful when we need to convert more than one unit in a given quantity. For example, suppose a flowmeter at a wastewater treatment facility gave us a flow measurement of 205 cubic feet per minute but we needed to convert this expression of water flow into units of cubic yards per day. Observe the following unit-fraction conversion to see how unity fractions serve the purpose of converting cubic feet into cubic yards, and minutes into days (by way of minutes to hours, and hours to days):

$$\left(\frac{205 \text{ } \cancel{\text{ft}}^3}{\cancel{\text{min}}}\right)\left(\frac{1^3 \text{ yd}^3}{3^3 \text{ } \cancel{\text{ft}}^3}\right)\left(\frac{60 \text{ } \cancel{\text{min}}}{1 \text{ } \cancel{\text{hr}}}\right)\left(\frac{24 \text{ } \cancel{\text{hr}}}{1 \text{ day}}\right) = 10933.3 \text{ yd}^3/\text{day}$$

Note how the only units left un-canceled on the left-hand side of the "equals" symbol are cubic yards (yd^3) and days, which therefore become the units of measurement for the final result.

A major caveat to this method of converting units is that the units must be *directly proportional* to one another, since this multiplicative conversion method is really nothing more than an exercise in mathematical proportions. Here are some examples (but not an exhaustive list!) of conversions that *cannot* be performed using the "unity fraction" method:

- Absolute / Gauge pressures, because one scale is *offset* from the other by 14.7 PSI (atmospheric pressure).
- Celsius / Fahrenheit, because one scale is *offset* from the other by 32 degrees.
- Wire diameter / gauge number, because gauge numbers grow smaller as wire diameter grows larger (inverse proportion rather than direct) and because there is no proportion relating the two.
- Power / decibels, because the relationship is logarithmic rather than proportional.

The following subsections give sets of physically equal quantities, which may be used to create unity fractions for unit conversion problems. Note that only those quantities shown in the same line (separated by = symbols) are truly equal to each other, not quantities appearing in different lines!

2.7 Percentages, per-unit, ppm, and ppb

In many circumstances it is useful to express a proportionality in terms of a special kind of fraction with a standardized denominator. A very common form of this is to represent a proportion as some fraction of one hundred, in which case we call it a *percentage*. For example, 50 percent (50%) means a proportion of $\frac{50}{100}$. "Percent" literally means "per cent" or "per hundred".

Earth's atmosphere, for example, contains approximately 20.9% oxygen gas by volume. This means that for every 100 molecules found in a sample of air, almost 21 of those are oxygen molecules. For extremely small proportions, however, percent becomes an awkward unit of measurement. In such cases it is common to see low concentrations of solute expressed as *parts per million* (ppm) or even *parts per billion* (ppb). The volumetric concentration of methane gas in Earth's atmosphere is a good example where parts-per-million is a more appropriate expression than percent: for every million molecules found in a sample of air, approximately 2 of them are methane molecules (i.e. methane has an atmospheric concentration of 2 ppm). As a percentage, this equates to only 0.0002%.

Another type of proportionality known as *per-unit* is used widely in the electrical power industry. Per-unit is simply a ratio of the given quantity to a base-unit value, and is equal to percentage divided by 100. For example, 50% is equivalent to 0.5 per-unit.

To use an example from an electric power system, suppose the measured voltage along a "230 kV" transmission line is actually 228.7 kV. We could say that the per-unit voltage of that line is 0.99435, since that is the value of the ratio $\frac{228.7}{230}$. Per-unit quantities simplify calculations within power systems because those per-unit quantities remain unaffected by other mathematical ratios in the system such as transformer ratios, $\sqrt{3}$ ratios for three-phase circuits, etc. For example, if that same 228.7 kV line voltage energized a 2:1 step-down transformer to output precisely half that amount of voltage (114.35 kV), that output voltage would have the same per-unit value of 0.99435 because that's also the value of the ratio $\frac{114.35}{115}$. Thus, a power engineer or technician calculating all the effects of a given voltage value in the system can work with the one per-unit value throughout the system rather than have to convert whenever the voltage in question gets stepped up or stepped down by transformers.

2.8 Scientific notation

Our standard way of representing numerical quantities is the *decimal system* where we limit ourselves to combinations of ten different characters (0 through 9) as well as a decimal point. A limitation of the decimal system is that these quantities become clumsy to write, read, and say when they become very large or very small.

For example, Avogadro's number is the number of carbon atoms[2] in a sample of pure ${}_{12}\mathrm{C}$ having a mass of 12 grams:

$$602,214,179,000,000,000,000,000$$

An example of an extremely small quantity is Boltzmann's constant which appears many times in the study of thermodynamics, which is even more challenging to accurately read because we don't use commas as separators like we do for large numbers with lots of trailing zeros:

$$.00000000000000000000013806504$$

One way to simplify the expression of such large and small quantities is to write them as values between one and ten multiplied by some power of ten. The purpose of that power-of-ten multiplier is to succinctly specify the placement of the decimal point, rather than using many zeros as place-holding digits. In other words, *the power tells you how many places to shift the decimal point.* For example, consider Avogadro's number and Boltzmann's constant re-written in this format:

$$6.02214179 \times 10^{23}$$

$$1.3806504 \times 10^{-23}$$

This is called *scientific notation*, and it is so popular that most computer programming languages and software applications accepting number values have some means of expressing it. For example, in the C, C++, and Python programming languages we could write Avogadro's number as `6.02214179E23`, the letter "E" representing $\times 10$.

[2] Or more generally, the number of entities in a *mole*. Think of a "mole" as being an extremely large half-dozen!

2.9 Metric prefixes

A similar approach for representing numerical values spanning very wide ranges is that used within the metric system of measurements, where specific powers of ten are represented by prefixes prepended to units of measurement. A table showing common metric prefixes appears here:

Prefix	Symbol	Power-of-ten
Exa	E	$10^{18} = 1,000,000,000,000,000,000$
Peta	P	$10^{15} = 1,000,000,000,000,000$
Tera	T	$10^{12} = 1,000,000,000,000$
Giga	G	$10^{9} = 1,000,000,000$
Mega	M	$10^{6} = 1,000,000$
Kilo	k	$10^{3} = 1,000$
Deka	da	$10^{1} = 10$
Deci	d	$10^{-1} = \frac{1}{10}$
Centi	c	$10^{-2} = \frac{1}{100}$
Milli	m	$10^{-3} = \frac{1}{1,000}$
Micro	μ	$10^{-6} = \frac{1}{1,000,000}$
Nano	n	$10^{-9} = \frac{1}{1,000,000,000}$
Pico	p	$10^{-12} = \frac{1}{1,000,000,000,000}$
Femto	f	$10^{-15} = \frac{1}{1,000,000,000,000,000}$
Atto	a	$10^{-18} = \frac{1}{1,000,000,000,000,000,000}$

In order to represent very large or very small quantities all we need to do is combine the right prefix with the unit. For example, a distance of 12,509 meters could be written as 1.2509×10^4 meters using scientific notation, or as 12.509 *kilo*meters (12.509 km) using the closest metric prefix. Similarly, a mass of 0.000000421 grams could be written as 0.421 *micro*grams (0.421 μg) or 421 *nano*grams (421 ng).

Just as computer programming languages and mathematical software applications generally support scientific notation both for entry and display, electronic hand calculators additionally offer a display mode called *engineering mode* where the standard scientific notation display (e.g. `1.2509E4`) is slightly modified such that all powers are integer multiples of three (e.g. `12.509E3`). Once you have committed to memory the power-of-ten associated with each common metric prefix (Mega through Pico is usually sufficient for most electronics work) it becomes a simple matter to translate the power-of-ten shown by a calculator in engineering mode to the corresponding metric prefix.

2.10 Quantifying error and tolerance

An important expression in most areas of scientific study is *error*, which is a way of stating how far off the expected mark a certain measurement or prediction is from its ideal value. This is usually done using percent, or for highly precise applications *ppm* or *ppb*.

Error is typically calculated as a ratio of the difference between the actual (measured or predicted) value and the ideal (theoretical) value divided by that same ideal value:

$$\text{Error (percent)} = \frac{\text{Actual} - \text{Theoretical}}{\text{Theoretical}} \times 100\%$$

$$\text{Error (ppm)} = \frac{\text{Actual} - \text{Theoretical}}{\text{Theoretical}} \times 10^6$$

$$\text{Error (ppb)} = \frac{\text{Actual} - \text{Theoretical}}{\text{Theoretical}} \times 10^9$$

For example, if a resistor is labeled as having a value of 2700 Ohms but after testing it we find it actually has a resistance of 2580 Ohms, we could calculate the error as:

$$\frac{2580 - 2700}{2700} \times 100\% = -4.444\%$$

$$\frac{2580 - 2700}{2700} \times 10^6 = -444,444.4 \text{ ppm}$$

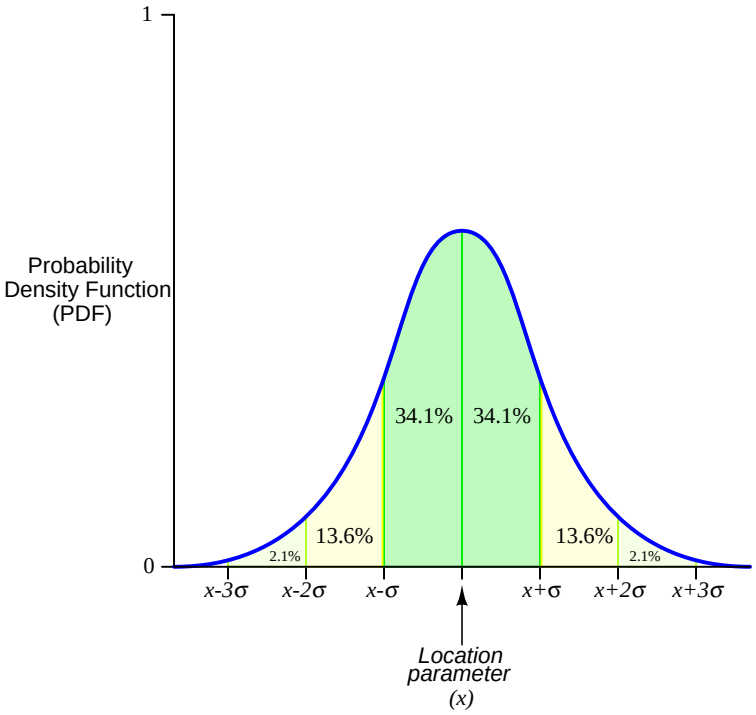
$$\frac{2580 - 2700}{2700} \times 10^9 = -444,444,444.4 \text{ ppb}$$

The mathematical sign of the calculated error is significant, as a negative value tells you the measured or predicted value is less than expected, while a positive value tells you it's more than expected.

Perfection is an unrealistic goal in any discipline of engineering, and so we commonly find components and systems alike rated with specified margins of allowable error called *tolerance*. The resistor error we just calculated was a little greater than -4%, but if we knew the resistor had a tolerance of $\pm$ 5% we would say that error was "within tolerance" and that the component is performing as rated.

2.11 Measurement uncertainty

Real-world measurements are always affected to some degree by noise and other random influences. This means, among other things, that measuring the same quantity multiple times often yields slightly different results, but that the majority of those measurements lie near some central value. Therefore, ultra-precise measurements are always given in terms of both that most-likely value (called the *location parameter*) as well as the *uncertainty* of the distribution, the latter always being a positive quantity. Uncertainty is typically given as the value of one standard deviation (σ) for the bell-shaped "Gaussian" curve describing[3] the range of measured values possible given the random influences.



The area encompassed by this curve represents 100% of all possible measurement values given the influences of random factors such as noise. The central location parameter (x) value is that measured value having the highest probability of occurrence. The area encompassed beneath the curve from x to $x + \sigma$ (the domain of one standard deviation from center) covers approximately 34.1% of the possible measurement values, with another 34.1% of measured values encompassed from x to $x - \sigma$.

[3] Such a *distribution curve* is actually a type of *histogram*, where the height of the curve at any point is proportional to the number of identical measurements at that value on the horizontal axis.

A practical example will help illustrate how these terms apply. Suppose we measured the height of all employees working for a particular company, and found all the measured heights fell along this Gaussian curve with an x height of 165 centimeters and a standard deviation of 6 centimeters. The x value of 165 cm simply means that this particular height measurement was the most common among all the employees. In this case, the source of height deviation is largely due to variations between the bodies of different individuals rather than random errors or noise, but the same statistical principles apply. This would mean 34.1% of the people working at this company measured between 165 and 171 centimeters (from x to $x + \sigma$), while another 34.1% measured between 159 and 165 centimeters (from $x - \sigma$ to x).

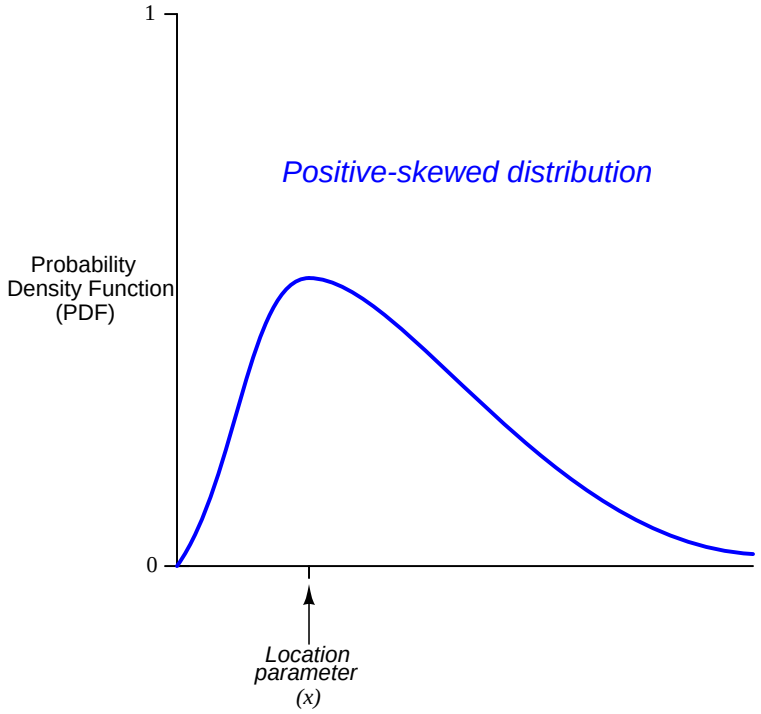
Applying this to physical constants, we may consult the *CODATA Recommended Values of the Fundamental Physical Constants* (NIST SP 961) published by the National Institute of Standards and Technology in May of 2019 for more examples. For instance, we find the measured mass ratio of one proton to one electron in this document has a value of 1836.15267343 with a standard deviation of 0.00000011. This means there is a 68.2% probability the true mass ratio value lies somewhere between 1836.15267332 and 1836.15267354 (i.e. from $x - \sigma$ to $x + \sigma$).

So far as anyone knows, all protons and all electrons are physically identical, and so the source of deviation must be random instrument errors and noise[4] rather than variations from one particle to another. It is also entirely possible (though unlikely) that the true mass ratio values is either higher than or lower than this $-\sigma$ to $+\sigma$ uncertainty interval.

A common format for expressing a measured value and its uncertainty is to append the significant digits of its uncertainty (over one standard deviation) in parentheses to the central value. In the case of the proton/electron mass ratio, we see it published by the NIST as 1836.15267343(11), and we take those uncertainty digits to apply to the final digits of the base number. In other words, the "11" uncertainty digits add to or subtract from the "43" digits at the end of the base value, representing the $x + \sigma$ and $x - \sigma$ values, respectively.

[4] This suggests uncertainty values will likely decrease over time as scientific measurement technologies and techniques improve with continued research and development. Uncertainty can never reach zero for empirical measurements, however, due to the unavoidable existence of certain types of noise in the universe.

Probability distributions for real-world measurements do not always follow the symmetrical pattern shown by a Gaussian curve. Sometimes the sources of uncertainty are asymmetrical, causing the probability distribution to become "lopsided". The direction and degree of this "lopsidedness" is called *skew*. An example of a probability distribution with a positive skew is shown below:



A good example of a positively-skewed probability distribution for a real-world measurement is the measurement of electrical resistance using a two-wire ohmmeter, in which the major source of uncertainty is extra electrical resistance stemming from the meter's test lead wires, contact resistance between the probe tips and the specimen being measured, and contact resistance between the test lead plugs and the meter's sockets (jacks). Since electrical resistances always *add* when in series with each other, these unknown resistances must always work to offset the measurement in a positive direction away from the true resistance of the specimen being measured. Other sources of error such as electrical noise, meter calibration error, and parallax error (only with analog meters) may influence the measured value either positively or negatively, but stray resistance in the test apparatus will always bias the measurement positively, thus skewing the probability distribution to the right of the highest probability (the peak location parameter).

2.12 Significant figures

In practically every science course one of the introductory topics is *significant figures*, presented along with a set of rules for truncating numerical results based on their uncertainty. These rules have a germ of truth to them, but they are actually quite misleading and conceptually confusing with regard to the nature of measurement uncertainty and how to practically deal with it. What I am about to say about significant figures will no doubt be controversial to some readers. I would challenge any of these readers to investigate metrology and notice how these commonly-taught "significant figures" are unknown to this field where uncertainty is of the utmost importance.

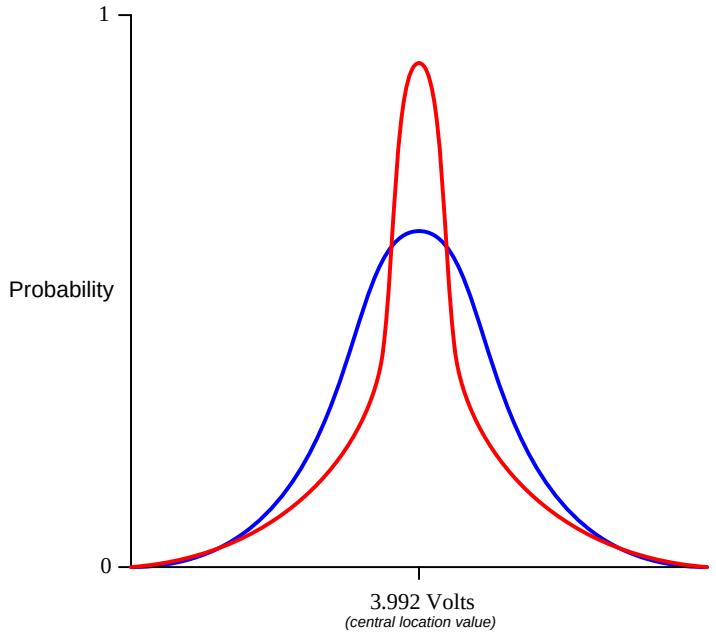
A "significant figure" in any numerical representation is a decimal digit necessary to the precision of that value. For example, the number 725 has three significant figures; the number 24000 has two significant figures (the "24" portion); the number 0.00029 has only two significant figures as well (the "29" portion); the number 63002 has five significant figures; etc. Place-holder zeroes representing powers of ten aren't "significant" in terms of precision because all they do is scale the magnitude of the number. One easy way to identify significant figures is to express the number in scientific notation and count the total digits to the left of the power-of-ten:

- $725 = 7.25 \times 10^2$ = three significant figures (the 7, 2, and 5)
- $24000 = 2.4 \times 10^4$ = two significant figures (the 2 and 4)
- $0.00029 = 2.9 \times 10^{-4}$ = two significant figures (the 2 and 9)
- $63002 = 6.3002 \times 10^4$ = five significant figures (the 6, 3, 0, 0, and 2)

The intended purpose of significant figures is to capture all the digits of a number that are meaningful insofar as we can accurately know that value. For example, if we measure the voltage across a circuit component using a voltmeter with a four-digit display, and that voltage measurement happens to be 3.992 Volts we would regard all four digits as being significant. However, if we measure a much smaller value with that exact same meter on the same range and the measurement happens to be 0.008 Volts we would only regard that value as having *one* significant figure (the "8") because the preceding zeroes do nothing but scale that measured value.

At first this advice would seem quite non-controversial, as it keeps us mindful of the limitations of our measurement instruments and of our measurement techniques. However, as we saw in the previous section a more sophisticated (and realistic!) assessment of measurement uncertainty means not all of the displayed digits are necessarily *certain*. For example, if our voltage measurement had an associated uncertainty of ± 0.014 Volts it would mean the 3.992 Volt measurement represents a possible variance of 3.978 Volts to 4.006 Volts over our standard confidence interval. This measurement uncertainty figure of ± 0.014 Volts cannot be determined by the number of displayed digits or the placement of significant figures within its display, but rather is a function of its design, component tolerances, external noise, and our measurement techniques.

A simple visual will help illustrate just one way in which the concept of significant figures is not the same as the concept of uncertainty. Take the following two distribution curves for voltage measurements, both curves having the exact same central location parameter value but having very different standard deviation (σ) values:



The red curve indicates the distribution of voltage measurements in a system having less uncertainty than the blue curve, despite the fact that both curves share the exact same center-point value of 3.992 Volts. In other words, there is a higher probability of the red system's actual voltage being 3.992 Volts than in the blue system. If both 3.992 Volt measurements were truly taken from a large sampling of individual voltage measurements to account for noise and other uncertainty sources, we can rest assured that all four digits of "3.992" are truly *significant* (i.e. important) to the measurement, yet the actual uncertainty of the true value is quite different between the blue curve and the red curve, the blue being the less *certain* of the two.

This is just one example, and not the only one, of how significance and uncertainty are distinctly different concepts. However, the standard approach to "significant figures" taught in many introductory science courses conflates these two concepts, attempting to employ the number of displayed digits as an expression expression of real-world significance *and* uncertainty.

Rules commonly associated with significant figures are even more problematic. These rules tell us to discard the significant figures of some measurements when we arithmetically combine them with other measured values. For example, it is common to be told that when adding or subtracting two or more numerical values, the final result should be truncated so as to have the same number of digits to the right of the decimal point as the measurement having the least of these. So, 725 plus 0.00029 would simply be 725. Multiplication and division follow a similar rule in that the result must contain the same number of significant figures as the measurement having the least of

these. So, 725 times 0.00029 would be 0.21. Like significant figures themselves, these arithmetic rules have a germ of truth to them – namely, that uncertainties propagate from measurement(s) to calculated result(s) – but unfortunately are too crude to properly treat the actual uncertainties of the measurements involved. We literally have no idea what the uncertainty of the "725" measurement is, nor of the "0.00029" measurement, based on those values alone, and so we really cannot tell from this data alone how many digits are truly "significant" in the computed results. Furthermore, if we did actually have uncertainty data on those two measurements we would not follow such crude rules as truncating entire digits (which at best is an order-of-magnitude *estimation* of uncertainty), and we would actually want to express the results' uncertainty in the same manner to the measurements': namely, as ± confidence intervals.

In summary, significant figures and their associated "rules" are half-truths unbecoming of their intended applications. In cases where the associated uncertainties are too small to matter to us, these rules only complicate. In cases where the uncertainties actually matter, these rules are far too simple to properly handle them. The cynic in me wonders if the prevalence of significant figures in science coursework is really just a means to force students to submit answers the same way to make grading easier!

For the types of measurements and calculations my students (who are to be electronics technicians) work with, I offer the following rules:

- Always record and enter into calculations *all* digits of any measurement. Never discard data, ever!! This is a basic principle of scientific ethics, that we preserve every bit of data collected so that others reviewing our data have opportunity to make discoveries we may have overlooked. Just because *we* think digits might be insignificant does not necessarily mean they are, and by recording all digits in full we permit others to discover phenomena we may have missed.

- All calculated results should be stored in the calculator/computer at full precision, never intentionally truncating digits for any reason. Any digits maintained within the digital calculator/computer above and beyond that necessary to capture the value and its uncertainty are called *guard digits*, and can only help (not hurt) accuracy. Storing calculated results and then recalling from the calculator/computer memory when performing subsequent calculations also carries with it the benefit of completely avoiding keystroke errors on the part of the user, as well as completely avoiding compounded errors resulting from repeated truncation of results as one result is used to calculate another!

- When hand-writing, typing, or otherwise displaying the final computed result(s), never round off those figures to a point where we will be unable to calculate deviation (error) to a degree at least as good as what is necessary for the application. For example, if our application requires an accuracy of ± 1%, three significant figures would be the minimum to express a value to within that tolerance (i.e. one part in one hundred) but four or more would be better. There is no limit to how *many* digits to record except convenience and readability, only a limit to how *few* you should record for the sake of not introducing rounding error.

2.13 Functions, tables, and graphs

In mathematics, a *function* is any process whereby one or more input values correspond to single output values. A logarithm is a good example of a function, where a single input value (the "argument") results in a single corresponding output value. Functions may take in multiple variables, but always output a single result for any given input values or combination of input values.

The input variable(s) to a function is/are called *independent*, while the output variable is called *dependent*. The concept here is that the output *depends on* the input(s), which in turn are arbitrary to the function. The span of possible values for the independent variable(s) is called the *domain* while the span of possible output values is called the *range*.

Mathematical functions appear throughout computer programming, always taking the form of an identifying label followed by a set of parentheses enclosing the independent variables (generally called the "arguments" to that function). For example, here is the common logarithm function being applied in the Python programming language, taking in an argument of 10000 and outputting a result of 4:

```
>>> log10(10000)
4.0
```

Here is another Python programming example, this one being the "power" function where the base and exponent are the two independent variables (i.e. arguments) to the function:

```
>>> pow(3,4)
81.0
```

A generic mathematical symbol for any function is the letter f followed by parentheses enclosing the independent variable(s). For example, a function taking in just one argument could be written as $f(x)$, and a function taking in two arguments could be written as $f(x, y)$. In the case of logarithms we could say $f(x) = \log(x)$. In the case of the power function we could say $f(b, x) = b^x$.

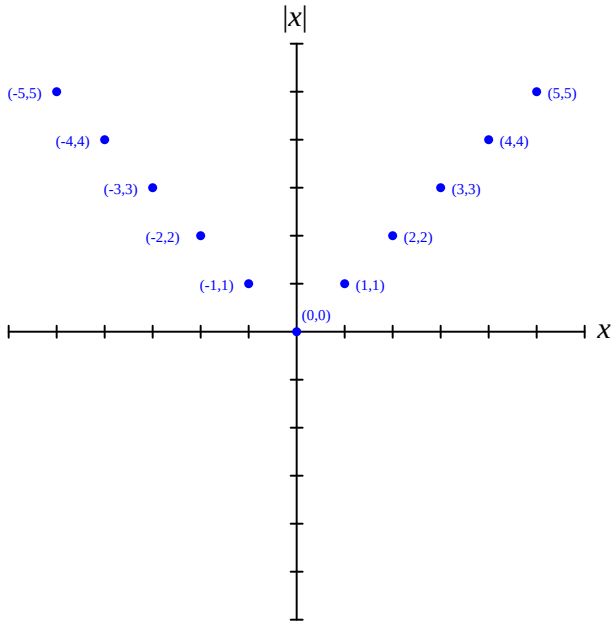
A great many physical phenomena may be represented either precisely or approximately by mathematical functions which is why functions are so critically important to the field of engineering.

One way to represent a function using specific numerical values is in the form of a *table*, with each variable (whether dependent or independent) represented by its own column in that table. For example, we may create a simple table showing sample input and output values for the *absolute number* function, the independent variable being represented by the symbol x and the dependent variable by $|x|$:

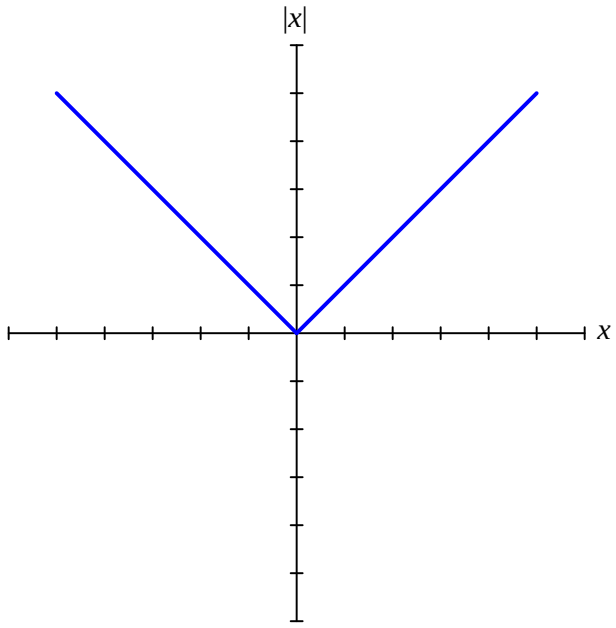
x	$\|x\|$
-5	5
-4	4
-3	3
-2	2
-1	1
0	0
1	1
2	2
3	3
4	4
5	5

A major limitation of tables is that they do not represent *all* possibilities of the function, because it is limited to the number of rows in the table. Here we see a domain of just -5 to 5 and a range of 0 to 5, with all values being integers. However, it is possible to apply the absolute value function to more than just these (e.g. $|-3.7| = 3.7$, $|234| = 234$, etc.).

Another way to numerically represent a function is in the form of a *graph*, which is a collection of dots where each dot is placed according to its alignment with multiple axes. The most common form of graph is the *Cartesian coordinate* graph which takes a two-dimensional rectangular form. By convention, the graph's horizontal axis is a number line for a single independent variable while the vertical axis is a number line for the dependent variable. For example, we may show a Cartesian coordinate graph for the same absolute value function previously tabulated, over a domain of −5 to 5:



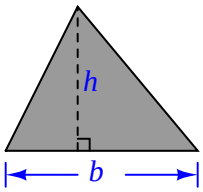
If we imagine plotting this same absolute value function for *all* real-number values within the domain of -5 to 5, what we will have is an *infinite number of dots* placed adjacent to each other, forming solid lines:



This is what most graphs look like: unbroken lines or curves tracing some shape on a two-dimensional grid. It's important to realize, though, that what appears to be an unbroken line or curve is just a very densely-spaced collection of individual points representing combinations of values satisfying the function.

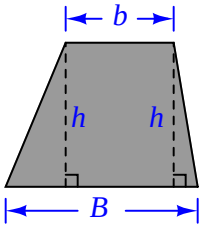
2.14 Common geometric shapes

Triangle



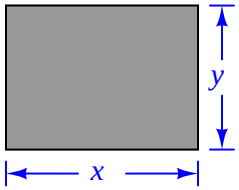
Area $A = \frac{1}{2}\,bh$

Trapezoid



Area $A = \frac{1}{2}\,(b + B)h$

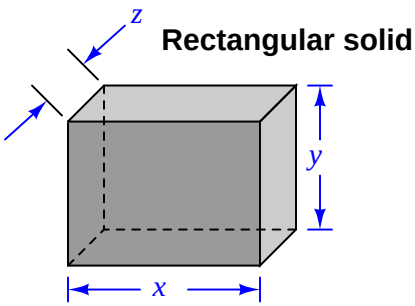
Rectangle



Perimeter $P = 2x + 2y$

Area $A = xy$

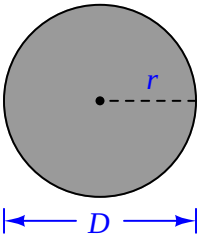
Rectangular solid



Surface area $A = 2xy + 2yz + 2xz$

Volume $V = xyz$

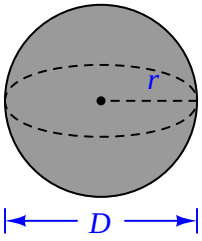
Circle



Circumference $C = \pi D = 2\pi r$

Area $A = \pi r^2$

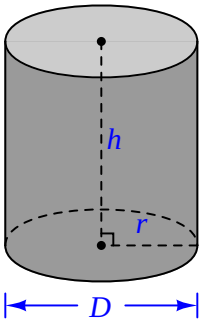
Sphere



Surface area $A = 4\pi r^2$

Volume $V = \frac{4}{3}\,\pi r^3$

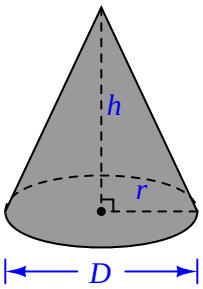
Right circular cylinder



Surface area $A = 2\pi r^2 + 2\pi rh$

Volume $V = \pi r^2 h$

Cone

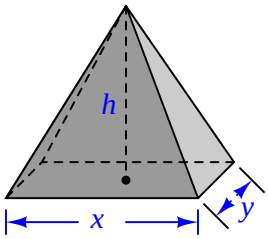


Surface area $A = \pi r \sqrt{r^2 + h^2} + \pi r^2$

Volume $V = \frac{1}{3} \pi r^2 h$

Note: the volume of any pyramid or cone is one-third the product of its height *(h)* and the area of its base.

Tetrahedron



Volume $V = \frac{1}{3} xyh$

One-dimensional measurements include height, width, depth, length, circumference, and perimeter; all of these measurements quantify *distance*. A simple way to conceptualize distance is to imagine measuring it with a length of string, a ruler, or a tape measure.

Two-dimensional measurements quantify *area*. A simple way to conceptualize area is to imagine how much paint would be required to cover that surface.

Three-dimensional measurements quantify *volume*. A simple way to conceptualize volume is to imagine how much liquid would be required to fill that space if the shape took the form of a vessel.

Chapter 3

Derivations and Technical References

This chapter is where you will find mathematical derivations too detailed to include in the tutorial, and/or tables and other technical reference material.

3.1 Resistor labeling

Resistors dissipate electrical energy in the form of heat, dropping voltage proportional to the amount of current passing through. This ratio of voltage to current is called *resistance* (R), measured in the unit of the *Ohm* (Ω), and it is the primary characterstic of any resistor. A popular method of labeling resistance values utilizes colored bands[1] to represent resistance values and tolerances. An example of this resistor style appears in the following photograph:

Four-band resistors are the most popular of the banded style, with the first and second bands representing significant digits, the third band representing a power-of-ten multiplier, and the fourth band representing tolerance.

Band color	**First digit value**	**Second digit value**	**Multiplier**	**Tolerance**
Black	0	0	10^0	
Brown	1	1	10^1	$\pm$ 1%
Red	2	2	10^2	$\pm$ 2%
Orange	3	3	10^3	$\pm$ 0.05%
Yellow	4	4	10^4	$\pm$ 0.02%
Green	5	5	10^5	$\pm$ 0.5%
Blue	6	6	10^6	$\pm$ 0.25%
Violet	7	7	10^7	$\pm$ 0.1%
Grey	8	8	10^8	$\pm$ 0.01%
White	9	9	10^9	
Gold			10^{-1}	$\pm$ 5%
Silver			10^{-2}	$\pm$ 10%
None				$\pm$ 20%

Applying this color code to the resistor shown above, we see that yellow = 4, violet = 7, red = 2, and silver = $\pm$ 10%. Therefore, this resistor's value is 47 $\times$ 10^2 Ohms (i.e. 4700 Ω) plus or minus 10%, which means it could be as low as 4230 Ω or as high as 5170 Ω.

[1] A useful mnemonic for associating these colors with decimal digits 0 through 9 and the percentages 5-10-20% is as follows: *"Better Be Right Or Your Great Big Venture Goes Wrong. Get Started Now."*.

Precision resistors require more significant digits to precisely quantify their values, and so use a *five-band* code where the first three bands represent digits, the fourth band represents the power-of-ten multiplier, and the fifth band represents tolerance. Both the four-band and five-band color codes are standardized as part of the international standard IEC 60062.

Many modern resistors are labeled with text rather than color-coded bands. For these resistors, a letter is used in place of a decimal point, the letter either being `R` (unit), `K` (kilo), `M` (mega), `G` (giga), `T` (tera), or `L` (milli). This is commonly referred to as the *RKM* code. Some labeling examples are shown in the following list:

- R82 = 0.82 Ω
- 3R9 = 3.9 Ω
- 47R = 47 Ω
- 560R = 560 Ω
- 3K3 = 3.3 kΩ
- 27K = 27 kΩ
- 100K = 100 kΩ
- 2M2 = 2.2 MΩ
- 15M = 15 MΩ
- 2L7 = 2.7 mΩ

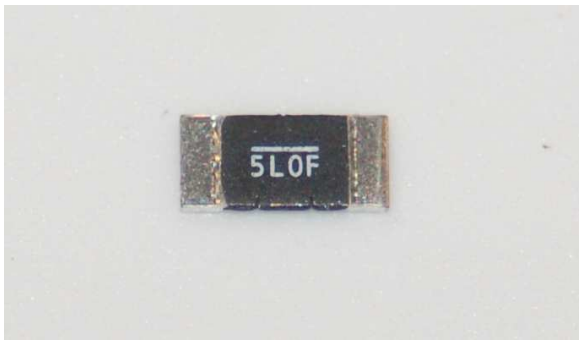
Tolerances for text-labeled resistors use the following letter codes, the tolerance letter always being the last character of the resistor's text label:

Letter code	**Tolerance**
B	± 0.1%
C	± 0.25%
D	± 0.5%
F	± 1%
G	± 2%
H	± 3%
J	± 5%
K	± 10%
M	± 20%
N	± 30%

For example, a resistor labeled `6K8J` would be 6.8 kΩ ± 5%.

Surface-mount device (SMD) resistors are often labeled with numbers only, either a three-digit numerical code or a four-digit numerical code. These codes follow the same pattern as four- and five-band color codes (except with no specified tolerance, just **digit-digit-multiplier** or **digit-digit-digit-multiplier**). For example, `473` would be 47 kΩ and `2711` would be 2.71 kΩ.

An exception to this general rule appears in the following photograph of a 5 milli-Ohm surface-mount resistor with a tolerance of ± 1%. The `5L0` portion of the code indicates 5.0 mΩ, while the `F` portion indicates the ±1% tolerance:

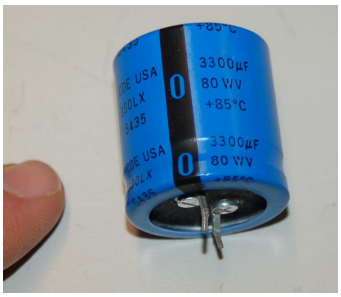


This particular SMD resistor happens to be useful as a high-current *shunt resistor* developing a precise voltage drop proportional to current. Its low resistance value does little to impede the flow of charge carriers to the load, while its tight tolerance ensures its voltage drop will be a fairly good representation of current through it.

3.2 Capacitor labeling

Capacitors store electrical energy using electric fields, the amount of energy stored being proportional to the square of the applied voltage. The ratio of stored energy to voltage (squared) is called *capacitance* (C), measured in the unit of the *Farad* (F), and it is the primary characterstic of any capacitor. Unfortunately, the labeling of capacitance on many capacitors is rather confusing. Color codes used to be popular for denoting capacitance value, following a very similar pattern to resistors, but numerical labels are now the norm. However, these labels tend to be rather terse, and this can cause interpretational difficulties.

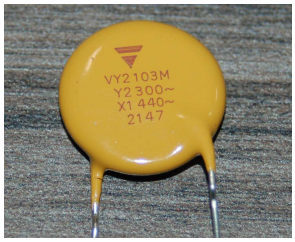
Physically large capacitors having sufficient surface area to allow many characters of printed text are the best in this regard.



The "supercapacitor" seen in the left-hand photograph is the clearest of them all: 650 Farads of capacitance with a working voltage rating of 2.7 Volts DC. The capacitor in the center photograph is also very unambiguous, having ratings of 3300 microFarads and 80 Volts DC ("WV" = *working voltage*). However the capacitor seen in the right-hand photograph, despite being physically larger than either of the others, is more cryptic: 16500 microFarads of capacitance with a working voltage rating of 50 Volts DC. Here we must simply assume that the "16500" figure refers to *micro*Farads.

Polarity is another important parameter for electrolytic-type capacitors such as these. For the supercapacitor (left photo) we see a stripe and letter "P" denoting the positive terminal of the capacitor; the 3300 microFarad capacitor (center photo) bears a black stripe and "−" symbol pointing toward its negative terminal; the 16500 microFarad capacitor (right photo) has polarity markings molded into the end-cap near its screw terminals (not visible in this photo).

Capacitors connected to AC line ("mains") voltage within line-powered appliances and other electronic devices must be *safety-rated*, as they are exposed to routine voltage transients such as those commonly experienced on electric power grids during lightning storms, and also because any short-circuit failure of a line-connected component would be catastrophic. For this reason, such capacitors often bear X and Y voltage ratings such as those seen on the following capacitor:



An X rating refers to the capacitor connected in parallel with the AC line's "hot" and "neutral" conductors: that is, directly across the line voltage. A Y rating refers to the capacitor connected between an Earth-potential conductor and either the "hot" or "neutral" conductor of an AC line source. These two different applications have their own safety concerns: a capacitor in an "X" application is at risk of catastrophically failing and possibly igniting nearby flammable materials if it develops an internal short-circuit, whereas a capacitor in a "Y" application is at risk of impressing dangerous electrical potential onto a metallic surface it internally fails shorted.

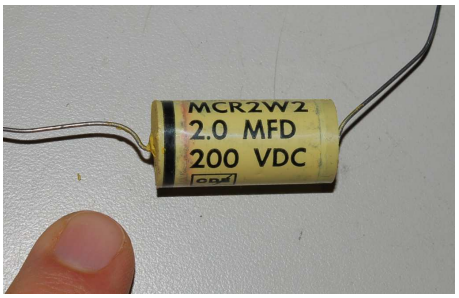
Immediately following the X or Y designator is a single numerical digit representing the rating sub-class, based on the peak impulse (transient) voltage:

Sub-class	**Maximum transient service voltage**
X1	± Between 2.5 kV and 4 kV
X2	± Less than 2.5 kV
X3	± Less than 1.2 kV

Sub-class	**Maximum transient test voltage**
Y1	± 8 kV
Y2	± 5 kV
Y3	± (none)
Y4	± 2.5 kV

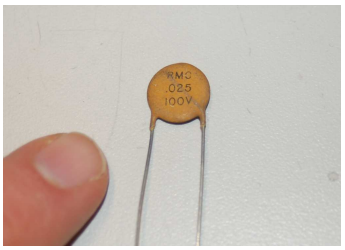
Any voltage value following an X or Y class/sub-class rating refers to the continuous AC voltage that capacitor is rated to handle in such a safety-critical application. For example, the `X1 440~` label on the capacitor shown in the photograph tells us that capacitor is rated for continuous duty across the poles of a 440 Volt AC source and that its service transient (impulse) rating is somewhere between 2500 Volts and 4000 Volts. This is the sort of capacitor we might find connected in parallel with the "hot" and "neutral" conductors of a 120 Volt AC power cord for suppression of high-frequency "noise".

Returning to our exploration of general capacitor labeling:



On the left we have a 2500 microFarad electrolytic capacitor rated for 50 Volts DC (positive on left), a capital letter "U" employed to symbolize *micro*. On the right we have a 2 microFarad plastic-film capacitor rated at 200 Volts DC, this time with the capital letter "M"[2] representing *micro*. Since plastic-film capacitors are non-polarized, the stripe on the left-hand side must represent something other than polarity: here it represents the terminal connected to the outer-most layer of metal foil[3] inside the capacitor.

Smaller capacitors become slightly more consistent, though no less confusing, in their labeling. Consider these examples:



From left to right we have (left) a 0.082 microFarad capacitor rated for 100 Volts DC[4], (center) a 0.025 microFarad capacitor rated for 100 Volts DC, and (right) a 0.68 microFarad capacitor rated for 160 Volts DC. The rule here is, any numerical capacitance value with a decimal point should be read as *micro*Farads. Note how the capacitor in the right-hand photograph uses a *comma* rather than a *point* between the "0" and the "68" because this one is of European manufacture, and it is conventional in European technical literature to use commas rather than points for decimal numbers.

[2] In the metric system a lower-case "m" is supposed to represent the prefix *milli* ($\times 10^{-3}$) and an upper-case "M" is supposed to represent the prefix *mega* ($\times 10^{6}$). However, since no technology yet invented is able to pack a *mega*Farad of capacitance into a single component, and for some unknown reason the prefix "milli" is never used for capacitance, we are expected to deduce that this letter "M" must represent *micro*. This, of course, flies in the face of standard metric notation, but it is nevertheless common to see in capacitor labeling.

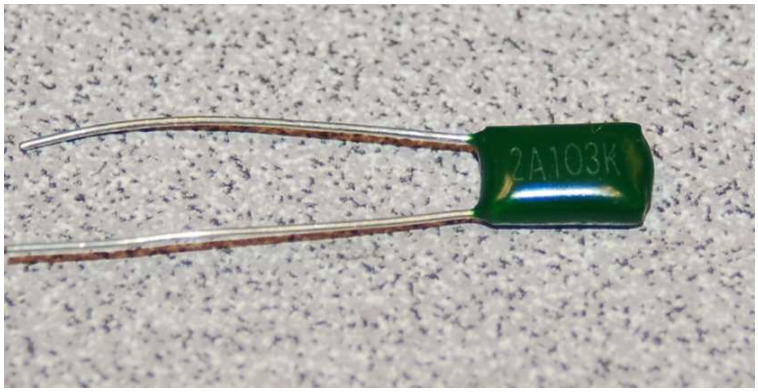
[3] This might be important to know for reasons of interference and signal coupling in densely-packed circuits. For a plastic-case capacitor such as this, the outer-most metal foil layer may comprise a parasitic capacitance with some adjacent component, and this in turn may degrade the circuit's performance. Best practice is to connect the "stripe" terminal of such a capacitor to ground or a power supply "rail" or some other node having a stable electrical potential.

[4] The second "0" of the number 100 happens to be partially rubbed off.

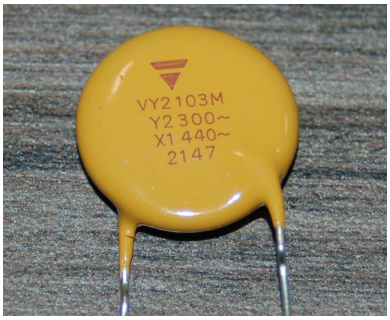
For capacitors with smaller capacitance values, the default metric prefix becomes *pico* ($\times 10^{-12}$) rather than micro. We can tell the difference between a "micro" capacitor and a "pico" capacitor by the presence or lack, respectively, of a decimal point. For example, a capacitor labeled 2.2 would be 2.2 microFarads in size, but a capacitor labeled 22 would be 22 picoFarads in size.

For decimal values of picoFarads, a lower-case letter "p" may be used as a decimal point; e.g. 2p2 would be 2.2 picoFarads. Similarly, the lower-case letter "n" may be used as a decimal point as well (e.g. 2n2 would be 2.2 nanoFarads). Confusingly, some manufacturers use a capital letter "R" as a generic decimal point for the capacitance value, the default metric prefix once again being *pico* (e.g. 2R2 would be 2.2 picoFarads).

A very common standard for small capacitor labeling is the use of three-digit numerical codes following the same digit-digit-multiplier format as resistor color codes, combined with the default assumption of *pico*Farads. For example, 332 would represent 33×10^2 picoFarads, which is equal to 3.3 nanoFarads. The following photograph shows a mylar film capacitor with a "103" code representing 10×10^3 picoFarads, which is equivalent to 10 nanoFarads:



Note the "2A" preceding the "103" capacitance rating: this is nothing but part of the manufacturer's part number. This is an unfortunate convention, of mixing manufacturer-specific part numbering with standardized value coding, and we see another example of it in the following photograph:



This Vishay capacitor also has a rating of 10 nanoFarads, but here we see the "103" capacitance code preceded by "YV2" which is unique to Vishay's part-numbering system. The last character

("M") specifies this capacitor's tolerance of ±20%, which is the next topic we will explore.

Capacitor *tolerance*, like resistor tolerance, specifies how far the particular device's capacitance may deviate from its advertised (nominal) value. In one of the previous examples, we saw a 0.082 microFarad capacitor with a ±10% tolerance, printed exactly as such on its body. However, for physically small capacitors where not enough surface area exists to print percentage figures, such as the 10 nF Vishay capacitor previously shown, we must use special letter codes to designate tolerance. These letter codes are also used for other components such as resistors, the following table showing several of them:

Letter code	**Tolerance**
A	± 0.05 pF
B	± 0.1 pF
C	± 0.25 pF
D	± 0.5 pF
E	± 0.5%
F	± 1%
G	± 2%
H	± 3%
J	± 5%
K	± 10%
L	± 15%
M	± 20%
N	± 30%
P	± −0% to +100%
S	± −20% to +50%
W	± −0% to +200%
X	± −20% to +40%
Z	± −20% to +80%

Note how the first four of these codes refer to absolute tolerances in picoFarads, while the rest represent tolerances in percent. When printed on a capacitor's body, the tolerance code typically follows the numerical capacitance code. For example, the photograph on the previous page showed a green-colored capacitor with "103K" on its body, which is 10×10^3 picoFarads ± 10%.

3.3 Inductor labeling

Inductors store electrical energy using magnetic fields, the amount of energy stored being proportional to the square of the applied current. The ratio of stored energy to current (squared) is called *inductance* (L), measured in the unit of the *Henry* (H), and it is the primary characterstic of any inductor. Color codes used to be popular for denoting inductance value, following a very similar pattern to resistors, but numerical labels are now the norm.

The color code used to mark inductors follows the same chromatic and numerical sequence used for resistors[5], with the first and second bands representing significant digits, the third band representing a power-of-ten multiplier, and the fourth band representing tolerance. However, rather than directly indicating inductance in Henrys, the implied metric prefix is *micro*.

Band color	**First digit value**	**Second digit value**	**Multiplier**	**Tolerance**
Black	0	0	10^0	± 20%
Brown	1	1	10^1	± 1%
Red	2	2	10^2	± 2%
Orange	3	3	10^3	± 3%
Yellow	4	4	10^4	± 4%
Green	5	5	10^5	
Blue	6	6	10^6	
Violet	7	7	10^7	
Grey	8	8	10^8	
White	9	9	10^9	
Gold			10^{-1}	± 5%
Silver			10^{-2}	± 10%
None				± 20%

For example, an inductor marked with colored bands Yellow, Violet, Orange, and Gold would be 47×10^3 microHenrys, or 47 mH.

Inductors labeled with numerical markings follow the same digit-digit-multiplier pattern as the color code: two significant digits followed by a power-of-ten multiplier, with an assumed base prefix of *micro*. For example, 331 would represent 33×10^1 microHenrys, which is 330 μH or 0.33 mH. If a decimal point is required for a numerically-labeled inductor, a capital letter "R" is used. For example, instead of labeling a 0.01 μH inductor as 0.01, it would be shown as R01.

[5] A useful mnemonic for associating these colors with decimal digits 0 through 9 and the percentages 5-10-20% is as follows: *"Better Be Right Or Your Great Big Venture Goes Wrong. Get Started Now."*.

Very small inductors are sometimes numerically labeled on the basis of *nano*Henrys rather than microHenrys. In such cases a capital letter "N" represents the decimal point for the nanoHenry value: for example, 3N9 represents 3.9 nH and 27N represents 27 nH.

Inductor *tolerance* is another important parameter which is often printed on the body of the inductor. For physically small inductors where not enough surface area exists to print percentage figures, we must resort to other means for expressing tolerance. To this end, a system of letter-codes has been developed, shown here in the following table:

Letter code	**Tolerance**
B	± 0.15 nH
C	± 0.2 nH
S	± 0.3 nH
D	± 0.5 nH
F	± 1%
G	± 2%
H	± 3%
J	± 5%
K	± 10%
L	± 15%
M	± 20%
V	± 25%
N	± 30%

Note how the first four of these codes refer to absolute tolerances in nanoHenrys, while the rest represent tolerances in percent. When printed on a inductor's body, the tolerance code typically follows the numerical inductance code. For example, an inductor with "821J" on its body would be 820 μH ± 5%.

3.4 IEC standard component values

Components such as resistors, inductors, and capacitors are manufactured in several *standard values*, described by IEC standard 60063. Rather than having a single series of standard values, the IEC publishes lists called *E series* based on the number of unique values spanning a single *decade* (i.e. a 10:1 range).

The shortest of these series, called *E3* contains just three values: 10, 22, and 47. The next series is called *E6* with six unique values: 10, 15, 22, 33, 47, and 68. These values represent *significant values* for components, meaning the decimal point may be freely moved to create values spanning multiple decades. For example, "33" simply means one can expect to find components manufactured in values of 33, 3.3, 0.33, and 0.033 as well as 330, 3.3 k, 33 k, etc.

Although this may seem like a strange standard for component manufacturers to follow, there is a compelling logic to it. The terms of each series are closer-spaced at the low end than at the high end, and this allows for *series* and/or *parallel* combinations of components to achieve most any desired value. For example, in the E6 series we only have values with the significant figures 10, 15, 22, 33, 47, and 68, but this doesn't mean we are limited to *total* values with these significant figures. For example, if we needed 80 Ohms of resistance we could connect a 33 Ohm and 47 Ohm resistor together in series. 50 Ohms could be made from two 68 Ohm resistors in parallel (making 34 Ohms) plus a 15 Ohm and 1 Ohm resistor in series.

On the next page is a table showing the four most common E-series specified by IEC standard 60063.

E3	**E6**	**E12**	**E24**
10	10	10	10
			11
		12	12
			13
	15	15	15
			16
		18	18
			20
22	22	22	22
			24
		27	27
			30
	33	33	33
			36
		39	39
			43
47	47	47	47
			51
		56	56
			62
	68	68	68
			75
		82	82
			91

E48, *E96*, and *E192* series are also found in the IEC 60063 standard, used for components with tighter tolerance ratings than typical.

Chapter 4

Programming References

A powerful tool for mathematical modeling is text-based *computer programming*. This is where you type coded commands in text form which the computer is able to interpret. Many different text-based languages exist for this purpose, but we will focus here on just two of them, *C++* and *Python*.

4.1 Programming in C++

One of the more popular text-based computer programming languages is called *C++*. This is a *compiled* language, which means you must create a plain-text file containing C++ code using a program called a *text editor*, then execute a software application called a *compiler* to translate your "source code" into instructions directly understandable to the computer. Here is an example of "source code" for a very simple C++ program intended to perform some basic arithmetic operations and print the results to the computer's console:

```
#include <iostream>
using namespace std;

int main (void)
{
  float x, y;

  x = 200;
  y = -560.5;

  cout << "This simple program performs basic arithmetic on" << endl;
  cout << "the two numbers " << x << " and " << y << " and then" << endl;
  cout << "displays the results on the computer's console." << endl;

  cout << endl;

  cout << "Sum = " << x + y << endl;
  cout << "Difference = " << x - y << endl;
  cout << "Product = " << x * y << endl;
  cout << "Quotient of " << x / y << endl;

  return 0;
}
```

Computer languages such as C++ are designed to make sense when read by human programmers. The general order of execution is left-to-right, top-to-bottom just the same as reading any text document written in English. Blank lines, indentation, and other "whitespace" is largely irrelevant in C++ code, and is included only to make the code more pleasing[1] to view.

[1] Although not included in this example, *comments* preceded by double-forward slash characters (`//`) may be added to source code as well to provide explanations of what the code is supposed to do, for the benefit of anyone reading it. The compiler application will ignore all comments.

Let's examine the C++ source code to explain what it means:

- `#include <iostream>` and `using namespace std;` are set-up instructions to the compiler giving it some context in which to interpret your code. The code specific to your task is located between the brace symbols ({ and }, often referred to as "curly-braces").
- `int main (void)` labels the "Main" function for the computer: the instructions within this function (lying between the { and } symbols) it will be commanded to execute. Every complete C++ program contains a `main` function at minimum, and often additional functions as well, but the `main` function is where execution always begins. The `int` declares this function will return an *integer* number value when complete, which helps to explain the purpose of the `return 0;` statement at the end of the `main` function: providing a numerical value of zero at the program's completion as promised by `int`. This returned value is rather incidental to our purpose here, but it is fairly standard practice in C++ programming.
- Grouping symbols such as (parentheses) and {braces} abound in C, C++, and other languages (e.g. Java). Parentheses typically group data to be processed by a function, called *arguments* to that function. Braces surround lines of executable code belonging to a particular function.
- The `float` declaration reserves places in the computer's memory for two *floating-point* variables, in this case the variables' names being `x` and `y`. In most text-based programming languages, variables may be named by single letters or by combinations of letters (e.g. `xyz` would be a single variable).
- The next two lines assign numerical values to the two variables. Note how each line terminates with a semicolon character (`;`) and how this pattern holds true for most of the lines in this program. In C++ semicolons are analogous to periods at the ends of English sentences. This demarcation of each line's end is necessary because C++ ignores whitespace on the page and doesn't "know" otherwise where one line ends and another begins.
- All the other instructions take the form of a `cout` command which prints characters to the "standard output" stream of the computer, which in this case will be text displayed on the console. The double-less-than symbols (`<<`) show data being sent *toward* the `cout` command. Note how verbatim text is enclosed in quotation marks, while variables such as `x` or mathematical expressions such as `x - y` are not enclosed in quotations because we want the computer to display the numerical values represented, not the literal text.
- Standard arithmetic operations (add, subtract, multiply, divide) are represented as `+`, `-`, `*`, and `/`, respectively.
- The `endl` found at the end of every `cout` statement marks the end of a line of text printed to the computer's console display. If not for these `endl` inclusions, the displayed text would resemble a run-on sentence rather than a paragraph. Note the `cout << endl;` line, which does nothing but create a blank line on the screen, for no reason other than esthetics.

After saving this *source code* text to a file with its own name (e.g. `myprogram.cpp`), you would then *compile* the source code into an *executable* file which the computer may then run. If you are using a console-based compiler such as *GCC* (very popular within variants of the Unix operating system[2], such as Linux and Apple's OS X), you would type the following command and press the Enter key:

```
g++ -o myprogram.exe myprogram.cpp
```

This command instructs the `GCC` compiler to take your source code (`myprogram.cpp`) and create with it an executable file named `myprogram.exe`. Simply typing `./myprogram.exe` at the command-line will then execute your program:

```
./myprogram.exe
```

If you are using a graphic-based C++ development system such as Microsoft Visual Studio[3], you may simply create a new console application "project" using this software, then paste or type your code into the example template appearing in the editor window, and finally run your application to test its output.

As this program runs, it displays the following text to the console:

```
This simple program performs basic arithmetic on
the two numbers 200 and -560.5 and then
displays the results on the computer's console.

Sum = -360.5
Difference = 760.5
Product = -112100
Quotient of -0.356824
```

As crude as this example program is, it serves the purpose of showing how easy it is to write and execute simple programs in a computer using the C++ language. As you encounter C++ example programs (shown as source code) in any of these modules, feel free to directly copy-and-paste the source code text into a text editor's screen, then follow the rest of the instructions given here (i.e. save to a file, compile, and finally run your program). You will find that it is generally easier to

[2] A very functional option for users of Microsoft Windows is called *Cygwin*, which provides a Unix-like console environment complete with all the customary utility applications such as GCC!

[3] Using Microsoft Visual Studio community version 2017 at the time of this writing to test this example, here are the steps I needed to follow in order to successfully compile and run a simple program such as this: (1) Start up Visual Studio and select the option to create a New Project; (2) Select the Windows Console Application template, as this will perform necessary set-up steps to generate a console-based program which will save you time and effort as well as avoid simple errors of omission; (3) When the editing screen appears, type or paste the C++ code within the `main()` function provided in the template, deleting the "Hello World" `cout` line that came with the template; (4) Type or paste any preprocessor directives (e.g. `#include` statements, `namespace` statements) necessary for your code that did not come with the template; (5) Lastly, under the Debug drop-down menu choose either Start Debugging (F5 hot-key) or Start Without Debugging (Ctrl-F5 hotkeys) to compile ("Build") and run your new program. Upon execution a console window will appear showing the output of your program.

learn computer programming by closely examining others' example programs and modifying them than it is to write your own programs starting from a blank screen.

4.2 Programming in Python

Another text-based computer programming language called *Python* allows you to type instructions at a terminal prompt and receive immediate results without having to compile that code. This is because Python is an *interpreted* language: a software application called an *interpreter* reads your source code, translates it into computer-understandable instructions, and then executes those instructions in one step.

The following shows what happens on my personal computer when I start up the Python interpreter on my personal computer, by typing `python3`[4] and pressing the Enter key:

```
Python 3.7.2 (default, Feb 19 2019, 18:15:18)
[GCC 4.1.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The `>>>` symbols represent the prompt within the Python interpreter "shell", signifying readiness to accept Python commands entered by the user.

Shown here is an example of the same arithmetic operations performed on the same quantities, using a Python interpreter. All lines shown preceded by the `>>>` prompt are entries typed by the human programmer, and all lines shown without the `>>>` prompt are responses from the Python interpreter software:

```
>>> x = 200
>>> y = -560.5
>>> x + y
-360.5
>>> x - y
760.5
>>> x * y
-112100.0
>>> x / y
-0.35682426404995538
>>> quit()
```

[4] Using version 3 of Python, which is the latest at the time of this writing.

More advanced mathematical functions are accessible in Python by first entering the line `from math import *` which "imports" these functions from Python's math *library* (with functions identical to those available for the C programming language, and included on any computer with Python installed). Some examples show some of these functions in use, demonstrating how the Python interpreter may be used as a scientific calculator:

```
>>> from math import *
>>> sin(30.0)
-0.98803162409286183
>>> sin(radians(30.0))
0.49999999999999994
>>> pow(2.0, 5.0)
32.0
>>> log10(10000.0)
4.0
>>> e
2.7182818284590451
>>> pi
3.1415926535897931
>>> log(pow(e,6.0))
6.0
>>> asin(0.7071068)
0.78539819000368838
>>> degrees(asin(0.7071068))
45.000001524425265
>>> quit()
```

Note how trigonometric functions assume angles expressed in *radians* rather than *degrees*, and how Python provides convenient functions for translating between the two. Logarithms assume a base of e unless otherwise stated (e.g. the `log10` function for common logarithms).

The interpreted (versus compiled) nature of Python, as well as its relatively simple syntax, makes it a good choice as a person's first programming language. For complex applications, interpreted languages such as Python execute slower than compiled languages such as C++, but for the very simple examples used in these learning modules speed is not a concern.

Another Python math library is `cmath`, giving Python the ability to perform arithmetic on complex numbers. This is very useful for AC circuit analysis using *phasors*[5] as shown in the following example. Here we see Python's interpreter used as a scientific calculator to show series and parallel impedances of a resistor, capacitor, and inductor in a 60 Hz AC circuit:

```
>>> from math import *
>>> from cmath import *
>>> r = complex(400,0)
>>> f = 60.0
>>> xc = 1/(2 * pi * f * 4.7e-6)
>>> zc = complex(0,-xc)
>>> xl = 2 * pi * f * 1.0
>>> zl = complex(0,xl)
>>> r + zc + zl
(400-187.38811239154882j)
>>> 1/(1/r + 1/zc + 1/zl)
(355.837695813625+125.35793777619385j)
>>> polar(r + zc + zl)
(441.717448903332, -0.4381072059213295)
>>> abs(r + zc + zl)
441.717448903332
>>> phase(r + zc + zl)
-0.4381072059213295
>>> degrees(phase(r + zc + zl))
-25.10169387356105
```

When entering a value in rectangular form, we use the `complex()` function where the arguments are the real and imaginary quantities, respectively. If we had opted to enter the impedance values in polar form, we would have used the `rect()` function where the first argument is the magnitude and the second argument is the angle in radians. For example, we could have set the capacitor's impedance (`zc`) as $X_C \angle -90^o$ with the command `zc = rect(xc,radians(-90))` rather than with the command `zc = complex(0,-xc)` and it would have worked the same.

Note how Python defaults to rectangular form for complex quantities. Here we defined a 400 Ohm resistance as a complex value in rectangular form (400 +j0 Ω), then computed capacitive and inductive reactances at 60 Hz and defined each of those as complex (phasor) values ($0 - jX_c$ Ω and $0 + jX_l$ Ω, respectively). After that we computed total impedance in series, then total impedance in parallel. Polar-form representation was then shown for the series impedance (441.717 Ω $\angle$ -25.102^o). Note the use of different functions to show the polar-form series impedance value: `polar()` takes the complex quantity and returns its polar magnitude and phase angle in *radians*; `abs()` returns just the polar magnitude; `phase()` returns just the polar angle, once again in radians. To find the polar phase angle in degrees, we nest the `degrees()` and `phase()` functions together.

The utility of Python's interpreter environment as a scientific calculator should be clear from these examples. Not only does it offer a powerful array of mathematical functions, but also unlimited

[5] A "phasor" is a voltage, current, or impedance represented as a complex number, either in rectangular or polar form.

assignment of variables as well as a convenient text record[6] of all calculations performed which may be easily copied and pasted into a text document for archival.

It is also possible to save a set of Python commands to a text file using a text editor application, and then instruct the Python interpreter to execute it at once rather than having to type it line-by-line in the interpreter's shell. For example, consider the following Python program, saved under the filename `myprogram.py`:

```
x = 200
y = -560.5

print("Sum")
print(x + y)

print("Difference")
print(x - y)

print("Product")
print(x * y)

print("Quotient")
print(x / y)
```

As with C++, the interpreter will read this source code from left-to-right, top-to-bottom, just the same as you or I would read a document written in English. Interestingly, whitespace *is* significant in the Python language (unlike C++), but this simple example program makes no use of that.

To execute this Python program, I would need to type `python myprogram.py` and then press the Enter key at my computer console's prompt, at which point it would display the following result:

```
Sum
-360.5
Difference
760.5
Product
-112100.0
Quotient
-0.35682426405
```

As you can see, syntax within the Python programming language is simpler than C++, which is one reason why it is often a preferred language for beginning programmers.

[6] Like many command-line computing environments, Python's interpreter supports "up-arrow" recall of previous entries. This allows quick recall of previously typed commands for editing and re-evaluation.

If you are interested in learning more about computer programming in *any* language, you will find a wide variety of books and free tutorials available on those subjects. Otherwise, feel free to learn by the examples presented in these modules.

Chapter 5

Questions

This learning module, along with all others in the ModEL collection, is designed to be used in an inverted instructional environment where students independently read[1] the tutorials and attempt to answer questions on their own *prior* to the instructor's interaction with them. In place of lecture[2], the instructor engages with students in Socratic-style dialogue, probing and challenging their understanding of the subject matter through inquiry.

Answers are not provided for questions within this chapter, and this is by design. Solved problems may be found in the Tutorial and Derivation chapters, instead. The goal here is *independence*, and this requires students to be challenged in ways where others cannot think for them. Remember that you always have the tools of *experimentation* and *computer simulation* (e.g. SPICE) to explore concepts!

The following lists contain ideas for Socratic-style questions and challenges. Upon inspection, one will notice a strong theme of *metacognition* within these statements: they are designed to foster a regular habit of examining one's own thoughts as a means toward clearer thinking. As such these sample questions are useful both for instructor-led discussions as well as for self-study.

[1] Technical reading is an essential academic skill for any technical practitioner to possess for the simple reason that the most comprehensive, accurate, and useful information to be found for developing technical competence is in textual form. Technical careers in general are characterized by the need for continuous learning to remain current with standards and technology, and therefore any technical practitioner who cannot read well is handicapped in their professional development. An excellent resource for educators on improving students' reading prowess through intentional effort and strategy is the book textitReading For Understanding – How Reading Apprenticeship Improves Disciplinary Learning in Secondary and College Classrooms by Ruth Schoenbach, Cynthia Greenleaf, and Lynn Murphy.

[2] Lecture is popular as a teaching method because it is easy to implement: any reasonably articulate subject matter expert can talk to students, even with little preparation. However, it is also quite problematic. A good lecture always makes complicated concepts seem easier than they are, which is bad for students because it instills a false sense of confidence in their own understanding; reading and re-articulation requires more cognitive effort and serves to verify comprehension. A culture of teaching-by-lecture fosters a debilitating dependence upon direct personal instruction, whereas the challenges of modern life demand independent and critical thought made possible only by gathering information and perspectives from afar. Information presented in a lecture is ephemeral, easily lost to failures of memory and dictation; text is forever, and may be referenced at any time.

General challenges following tutorial reading

- Summarize as much of the text as you can in one paragraph of your own words. A helpful strategy is to explain ideas as you would for an intelligent child: as simple as you can without compromising too much accuracy.
- Simplify a particular section of the text, for example a paragraph or even a single sentence, so as to capture the same fundamental idea in fewer words.
- Where did the text make the most sense to you? What was it about the text's presentation that made it clear?
- Identify where it might be easy for someone to misunderstand the text, and explain why you think it could be confusing.
- Identify any new concept(s) presented in the text, and explain in your own words.
- Identify any familiar concept(s) such as physical laws or principles applied or referenced in the text.
- Devise a proof of concept experiment demonstrating an important principle, physical law, or technical innovation represented in the text.
- Devise an experiment to disprove a plausible misconception.
- Did the text reveal any misconceptions you might have harbored? If so, describe the misconception(s) and the reason(s) why you now know them to be incorrect.
- Describe any useful problem-solving strategies applied in the text.
- Devise a question of your own to challenge a reader's comprehension of the text.

General follow-up challenges for assigned problems

- Identify where any fundamental laws or principles apply to the solution of this problem, especially before applying any mathematical techniques.
- Devise a thought experiment to explore the characteristics of the problem scenario, applying known laws and principles to mentally model its behavior.
- Describe in detail your own strategy for solving this problem. How did you identify and organized the given information? Did you sketch any diagrams to help frame the problem?
- Is there more than one way to solve this problem? Which method seems best to you?
- Show the work you did in solving this problem, even if the solution is incomplete or incorrect.
- What would you say was the most challenging part of this problem, and why was it so?
- Was any important information missing from the problem which you had to research or recall?
- Was there any extraneous information presented within this problem? If so, what was it and why did it not matter?
- Examine someone else's solution to identify where they applied fundamental laws or principles.
- Simplify the problem from its given form and show how to solve this simpler version of it. Examples include eliminating certain variables or conditions, altering values to simpler (usually whole) numbers, applying a limiting case (i.e. altering a variable to some extreme or ultimate value).
- For quantitative problems, identify the real-world meaning of all intermediate calculations: their units of measurement, where they fit into the scenario at hand. Annotate any diagrams or illustrations with these calculated values.
- For quantitative problems, try approaching it qualitatively instead, thinking in terms of "increase" and "decrease" rather than definite values.
- For qualitative problems, try approaching it quantitatively instead, proposing simple numerical values for the variables.
- Were there any assumptions you made while solving this problem? Would your solution change if one of those assumptions were altered?
- Identify where it would be easy for someone to go astray in attempting to solve this problem.
- Formulate your own problem based on what you learned solving this one.

General follow-up challenges for experiments or projects

- In what way(s) was this experiment or project easy to complete?
- Identify some of the challenges you faced in completing this experiment or project.

- Show how thorough documentation assisted in the completion of this experiment or project.
- Which fundamental laws or principles are key to this system's function?
- Identify any way(s) in which one might obtain false or otherwise misleading measurements from test equipment in this system.
- What will happen if (component X) fails (open/shorted/etc.)?
- What would have to occur to make this system unsafe?

5.1 Conceptual reasoning

These questions are designed to stimulate your analytic and synthetic thinking[3]. In a Socratic discussion with your instructor, the goal is for these questions to prompt an extended dialogue where assumptions are revealed, conclusions are tested, and understanding is sharpened. Your instructor may also pose additional questions based on those assigned, in order to further probe and refine your conceptual understanding.

Questions that follow are presented to challenge and probe your understanding of various concepts presented in the tutorial. These questions are intended to serve as a guide for the Socratic dialogue between yourself and the instructor. Your instructor's task is to ensure you have a sound grasp of these concepts, and the questions contained in this document are merely a means to this end. Your instructor may, at his or her discretion, alter or substitute questions for the benefit of tailoring the discussion to each student's needs. The only absolute requirement is that each student is challenged and assessed at a level equal to or greater than that represented by the documented questions.

It is far more important that you convey your *reasoning* than it is to simply convey a correct answer. For this reason, you should refrain from researching other information sources to answer questions. What matters here is that *you* are doing the thinking. If the answer is incorrect, your instructor will work with you to correct it through proper reasoning. A correct answer without an adequate explanation of how you derived that answer is unacceptable, as it does not aid the learning or assessment process.

You will note a conspicuous lack of answers given for these conceptual questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. The best way by far is to debate the answers with fellow students and also with the instructor during the Socratic dialogue sessions intended to be used with these learning modules. Reasoning through challenging questions with other people is an excellent tool for developing strong reasoning skills.

Another means of checking your conceptual answers, where applicable, is to use circuit simulation software to explore the effects of changes made to circuits. For example, if one of these conceptual questions challenges you to predict the effects of altering some component parameter in a circuit, you may check the validity of your work by simulating that same parameter change within software and seeing if the results agree.

[3] *Analytical* thinking involves the "disassembly" of an idea into its constituent parts, analogous to dissection. *Synthetic* thinking involves the "assembly" of a new idea comprised of multiple concepts, analogous to construction. Both activities are high-level cognitive skills, extremely important for effective problem-solving, necessitating frequent challenge and regular practice to fully develop.

5.1.1 Reading outline and reflections

"Reading maketh a full man; conference a ready man; and writing an exact man" – Francis Bacon

Francis Bacon's advice is a blueprint for effective education: reading provides the learner with knowledge, writing focuses the learner's thoughts, and critical dialogue equips the learner to confidently communicate and apply their learning. Independent acquisition and application of knowledge is a powerful skill, well worth the effort to cultivate. To this end, students should read these educational resources closely, write their own outline and reflections on the reading, and discuss in detail their findings with classmates and instructor(s). You should be able to do all of the following after reading any instructional text:

☑ Briefly OUTLINE THE TEXT, as though you were writing a detailed Table of Contents. Feel free to rearrange the order if it makes more sense that way. Prepare to articulate these points in detail and to answer questions from your classmates and instructor. Outlining is a good self-test of thorough reading because you cannot outline what you have not read or do not comprehend.

☑ Demonstrate ACTIVE READING STRATEGIES, including verbalizing your impressions as you read, simplifying long passages to convey the same ideas using fewer words, annotating text and illustrations with your own interpretations, working through mathematical examples shown in the text, cross-referencing passages with relevant illustrations and/or other passages, identifying problem-solving strategies applied by the author, etc. Technical reading is a special case of problem-solving, and so these strategies work precisely because they help solve any problem: paying attention to your own thoughts (metacognition), eliminating unnecessary complexities, identifying what makes sense, paying close attention to details, drawing connections between separated facts, and noting the successful strategies of others.

☑ Identify IMPORTANT THEMES, especially GENERAL LAWS and PRINCIPLES, expounded in the text and express them in the simplest of terms as though you were teaching an intelligent child. This emphasizes connections between related topics and develops your ability to communicate complex ideas to anyone.

☑ Form YOUR OWN QUESTIONS based on the reading, and then pose them to your instructor and classmates for their consideration. Anticipate both correct and incorrect answers, the incorrect answer(s) assuming one or more plausible misconceptions. This helps you view the subject from different perspectives to grasp it more fully.

☑ Devise EXPERIMENTS to test claims presented in the reading, or to disprove misconceptions. Predict possible outcomes of these experiments, and evaluate their meanings: what result(s) would confirm, and what would constitute disproof? Running mental simulations and evaluating results is essential to scientific and diagnostic reasoning.

☑ Specifically identify any points you found CONFUSING. The reason for doing this is to help diagnose misconceptions and overcome barriers to learning.

5.1.2 Foundational concepts

Correct analysis and diagnosis of electric circuits begins with a proper understanding of some basic concepts. The following is a list of some important concepts referenced in this module's full tutorial. Define each of them in your own words, and be prepared to illustrate each of these concepts with a description of a practical example and/or a live demonstration.

Energy

Conservation of Energy

Simplification as a problem-solving strategy

Thought experiments as a problem-solving strategy

Limiting cases as a problem-solving strategy

Annotating diagrams as a problem-solving strategy

Interpreting intermediate results as a problem-solving strategy

Graphing as a problem-solving strategy

Converting a qualitative problem into a quantitative problem

Converting a quantitative problem into a qualitative problem

Working "backwards" to validate calculated results

Reductio ad absurdum

Re-drawing schematics as a problem-solving strategy

Cut-and-try problem-solving strategy

Algebraic substitution

???

5.1.3 First conceptual question

Challenges

- ???.
- ???.
- ???.

5.1.4 Second conceptual question

Challenges

- ???.
- ???.
- ???.

5.2 Quantitative reasoning

These questions are designed to stimulate your computational thinking. In a Socratic discussion with your instructor, the goal is for these questions to reveal your mathematical approach(es) to problem-solving so that good technique and sound reasoning may be reinforced. Your instructor may also pose additional questions based on those assigned, in order to observe your problem-solving firsthand.

Mental arithmetic and estimations are strongly encouraged for all calculations, because without these abilities you will be unable to readily detect errors caused by calculator misuse (e.g. keystroke errors).

You will note a conspicuous lack of answers given for these quantitative questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. My advice is to use circuit simulation software such as SPICE to check the correctness of quantitative answers. Refer to those learning modules within this collection focusing on SPICE to see worked examples which you may use directly as practice problems for your own study, and/or as templates you may modify to run your own analyses and generate your own practice problems.

Completely worked example problems found in the Tutorial may also serve as "test cases[4]" for gaining proficiency in the use of circuit simulation software, and then once that proficiency is gained you will never need to rely[5] on an answer key!

[4] In other words, set up the circuit simulation software to analyze the same circuit examples found in the Tutorial. If the simulated results match the answers shown in the Tutorial, it confirms the simulation has properly run. If the simulated results disagree with the Tutorial's answers, something has been set up incorrectly in the simulation software. Using every Tutorial as practice in this way will quickly develop proficiency in the use of circuit simulation software.

[5] This approach is perfectly in keeping with the instructional philosophy of these learning modules: *teaching students to be self-sufficient thinkers*. Answer keys can be useful, but it is even more useful to your long-term success to have a set of tools on hand for checking your own work, because once you have left school and are on your own, there will no longer be "answer keys" available for the problems you will have to solve.

5.2.1 Miscellaneous physical constants

Note: constants shown in **bold** type are *exact*, not approximations. Values inside of parentheses show one standard deviation (σ) of uncertainty in the final digits: for example, the magnetic permeability of free space value given as $1.25663706212(19) \times 10^{-6}$ H/m represents a center value (i.e. the location parameter) of $1.25663706212 \times 10^{-6}$ Henrys per meter with one standard deviation of uncertainty equal to $0.0000000000019 \times 10^{-6}$ Henrys per meter.

Avogadro's number (N_A) = **6.02214076** $\times 10^{23}$ **per mole** (mol^{-1})

Boltzmann's constant (k) = **1.380649** $\times 10^{-23}$ **Joules per Kelvin** (J/K)

Electronic charge (e) = **1.602176634** $\times 10^{-19}$ **Coulomb** (C)

Faraday constant (F) = **96,485.33212...** $\times 10^{4}$ **Coulombs per mole** (C/mol)

Magnetic permeability of free space (μ_0) = $1.25663706212(19) \times 10^{-6}$ Henrys per meter (H/m)

Electric permittivity of free space (ϵ_0) = $8.8541878128(13) \times 10^{-12}$ Farads per meter (F/m)

Characteristic impedance of free space (Z_0) = 376.730313668(57) Ohms (Ω)

Gravitational constant (G) = $6.67430(15) \times 10^{-11}$ cubic meters per kilogram-seconds squared ($m^3/kg\text{-}s^2$)

Molar gas constant (R) = **8.314462618... Joules per mole-Kelvin** (J/mol-K) = 0.08205746(14) liters-atmospheres per mole-Kelvin

Planck constant (h) = **6.62607015** $\times 10^{-34}$ **joule-seconds** (J-s)

Stefan-Boltzmann constant (σ) = **5.670374419...** $\times 10^{-8}$ **Watts per square meter-Kelvin**^{4} ($W/m^2 \cdot K^4$)

Speed of light in a vacuum (c) = **299,792,458 meters per second** (m/s) = 186282.4 miles per second (mi/s)

Note: All constants taken from NIST data "Fundamental Physical Constants – Complete Listing", from `http://physics.nist.gov/constants`, National Institute of Standards and Technology (NIST), 2018 CODATA Adjustment.

5.2.2 Introduction to spreadsheets

A powerful computational tool you are encouraged to use in your work is a *spreadsheet*. Available on most personal computers (e.g. Microsoft Excel), *spreadsheet* software performs numerical calculations based on number values and formulae entered into cells of a grid. This grid is typically arranged as lettered columns and numbered rows, with each cell of the grid identified by its column/row coordinates (e.g. cell `B3`, cell `A8`). Each cell may contain a string of text, a number value, or a mathematical formula. The spreadsheet automatically updates the results of all mathematical formulae whenever the entered number values are changed. This means it is possible to set up a spreadsheet to perform a series of calculations on entered data, and those calculations will be re-done by the computer any time the data points are edited in any way.

For example, the following spreadsheet calculates average speed based on entered values of distance traveled and time elapsed:

	A	B	C	D
1	Distance traveled	46.9	Kilometers	
2	Time elapsed	1.18	Hours	
3	Average speed	= B1 / B2	km/h	
4				
5				

Text labels contained in cells `A1` through `A3` and cells `C1` through `C3` exist solely for readability and are not involved in any calculations. Cell `B1` contains a sample distance value while cell `B2` contains a sample time value. The formula for computing speed is contained in cell `B3`. Note how this formula begins with an "equals" symbol (`=`), references the values for distance and speed by lettered column and numbered row coordinates (`B1` and `B2`), and uses a forward slash symbol for division (`/`). The coordinates `B1` and `B2` function as *variables*[6] would in an algebraic formula.

When this spreadsheet is executed, the numerical value `39.74576` will appear in cell `B3` rather than the formula `= B1 / B2`, because 39.74576 is the computed speed value given 46.9 kilometers traveled over a period of 1.18 hours. If a different numerical value for distance is entered into cell `B1` or a different value for time is entered into cell `B2`, cell `B3`'s value will automatically update. All you need to do is set up the given values and any formulae into the spreadsheet, and the computer will do all the calculations for you.

Cell `B3` may be referenced by other formulae in the spreadsheet if desired, since it is a variable just like the given values contained in `B1` and `B2`. This means it is possible to set up an entire chain of calculations, one dependent on the result of another, in order to arrive at a final value. The arrangement of the given data and formulae need not follow any pattern on the grid, which means you may place them anywhere.

[6] Spreadsheets may also provide means to attach text labels to cells for use as variable names (Microsoft Excel simply calls these labels "names"), but for simple spreadsheets such as those shown here it's usually easier just to use the standard coordinate naming for each cell.

Common[7] arithmetic operations available for your use in a spreadsheet include the following:

- Addition (`+`)
- Subtraction (`-`)
- Multiplication (`*`)
- Division (`/`)
- Powers (`^`)
- Square roots (`sqrt()`)
- Logarithms (`ln()` , `log10()`)

Parentheses may be used to ensure[8] proper order of operations within a complex formula. Consider this example of a spreadsheet implementing the *quadratic formula*, used to solve for roots of a polynomial expression in the form of $ax^2 + bx + c$:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

	A	B
1	x_1	= (-B4 + sqrt((B4^2) - (4*B3*B5))) / (2*B3)
2	x_2	= (-B4 - sqrt((B4^2) - (4*B3*B5))) / (2*B3)
3	a =	9
4	b =	5
5	c =	-2

This example is configured to compute roots[9] of the polynomial $9x^2 + 5x - 2$ because the values of 9, 5, and -2 have been inserted into cells `B3`, `B4`, and `B5`, respectively. Once this spreadsheet has been built, though, it may be used to calculate the roots of *any* second-degree polynomial expression simply by entering the new a, b, and c coefficients into cells `B3` through `B5`. The numerical values appearing in cells `B1` and `B2` will be automatically updated by the computer immediately following any changes made to the coefficients.

[7] Modern spreadsheet software offers a bewildering array of mathematical functions you may use in your computations. I recommend you consult the documentation for your particular spreadsheet for information on operations other than those listed here.

[8] Spreadsheet programs, like text-based programming languages, are designed to follow standard order of operations by default. However, my personal preference is to use parentheses even where strictly unnecessary just to make it clear to any other person viewing the formula what the intended order of operations is.

[9] Reviewing some algebra here, a *root* is a value for x that yields an overall value of zero for the polynomial. For this polynomial ($9x^2 + 5x - 2$) the two roots happen to be $x = 0.269381$ and $x = -0.82494$, with these values displayed in cells `B1` and `B2`, respectively upon execution of the spreadsheet.

Alternatively, one could break up the long quadratic formula into smaller pieces like this:

$$y = \sqrt{b^2 - 4ac} \qquad z = 2a$$

$$x = \frac{-b \pm y}{z}$$

	A	B	C
1	x_1	= (-B4 + C1) / C2	= sqrt((B4^2) - (4*B3*B5))
2	x_2	= (-B4 - C1) / C2	= 2*B3
3	a =	9	
4	b =	5	
5	c =	-2	

Note how the square-root term (y) is calculated in cell C1, and the denominator term (z) in cell C2. This makes the two final formulae (in cells B1 and B2) simpler to interpret. The positioning of all these cells on the grid is completely arbitrary[10] – all that matters is that they properly reference each other in the formulae.

Spreadsheets are particularly useful for situations where the same set of calculations representing a circuit or other system must be repeated for different initial conditions. The power of a spreadsheet is that it automates what would otherwise be a tedious set of calculations. One specific application of this is to simulate the effects of various components within a circuit failing with abnormal values (e.g. a shorted resistor simulated by making its value nearly zero; an open resistor simulated by making its value extremely large). Another application is analyzing the behavior of a circuit design given new components that are out of specification, and/or aging components experiencing drift over time.

[10] My personal preference is to locate all the "given" data in the upper-left cells of the spreadsheet grid (each data point flanked by a sensible name in the cell to the left and units of measurement in the cell to the right as illustrated in the first distance/time spreadsheet example), sometimes coloring them in order to clearly distinguish which cells contain entered data versus which cells contain computed results from formulae. I like to place all formulae in cells below the given data, and try to arrange them in logical order so that anyone examining my spreadsheet will be able to figure out *how* I constructed a solution. This is a general principle I believe all computer programmers should follow: *document and arrange your code to make it easy for other people to learn from it.*

5.2.3 Plain, metric, and scientific notations

Complete the following table with equivalent values in each row:

Plain	**Metric prefix**	**Scientific notation**
34,000 grams (g)	34 kilograms (kg)	3.4 $\times 10^{4}$ grams (g)
932,500 meters (m)		
		1.422 $\times 10^{3}$ Volts (V)
	0.328 milliAmperes (mA)	
		4.7 $\times 10^{-6}$ Farads (F)
0.0357 Henrys (H)		
		1.570814 $\times 10^{6}$ Hertz (Hz)
	10.7 microseconds (μs)	
		5.231 $\times 10^{-7}$ Coulombs (C)
	29.5 milliSiemens (mS)	
0.000097 calories (c)		

Challenges

- What purpose does scientific notation serve?
- Is there any advantage to expressing a quantity as 0.7325 milliVolts versus 732.5 microVolts?

5.2.4 Second quantitative problem

Challenges

- ???.
- ???.
- ???.

5.3 Diagnostic reasoning

These questions are designed to stimulate your deductive and inductive thinking, where you must apply general principles to specific scenarios (deductive) and also derive conclusions about the failed circuit from specific details (inductive). In a Socratic discussion with your instructor, the goal is for these questions to reinforce your recall and use of general circuit principles and also challenge your ability to integrate multiple symptoms into a sensible explanation of what's wrong in a circuit. Your instructor may also pose additional questions based on those assigned, in order to further challenge and sharpen your diagnostic abilities.

As always, your goal is to fully *explain* your analysis of each problem. Simply obtaining a correct answer is not good enough – you must also demonstrate sound reasoning in order to successfully complete the assignment. Your instructor's responsibility is to probe and challenge your understanding of the relevant principles and analytical processes in order to ensure you have a strong foundation upon which to build further understanding.

You will note a conspicuous lack of answers given for these diagnostic questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. The best way by far is to debate the answers with fellow students and also with the instructor during the Socratic dialogue sessions intended to be used with these learning modules. Reasoning through challenging questions with other people is an excellent tool for developing strong reasoning skills.

Another means of checking your diagnostic answers, where applicable, is to use circuit simulation software to explore the effects of faults placed in circuits. For example, if one of these diagnostic questions requires that you predict the effect of an open or a short in a circuit, you may check the validity of your work by simulating that same fault (substituting a very high resistance in place of that component for an open, and substituting a very low resistance for a short) within software and seeing if the results agree.

5.3.1 First diagnostic scenario

Challenges

- ???.
- ???.
- ???.

5.3.2 Second diagnostic scenario

Challenges

- ???.
- ???.
- ???.

Appendix A

Problem-Solving Strategies

The ability to solve complex problems is arguably one of the most valuable skills one can possess, and this skill is particularly important in any science-based discipline.

- Study principles, not procedures. Don't be satisfied with merely knowing how to compute solutions – learn *why* those solutions work.
- Identify what it is you need to solve, identify all relevant data, identify all units of measurement, identify any general principles or formulae linking the given information to the solution, and then identify any "missing pieces" to a solution. Annotate all diagrams with this data.
- Sketch a diagram to help visualize the problem. When building a real system, always devise a plan for that system and analyze its function *before* constructing it.
- Follow the units of measurement and meaning of every calculation. If you are ever performing mathematical calculations as part of a problem-solving procedure, and you find yourself unable to apply each and every intermediate result to some aspect of the problem, it means you don't understand what you are doing. Properly done, every mathematical result should have practical meaning for the problem, and not just be an abstract number. You should be able to identify the proper units of measurement for each and every calculated result, and show where that result fits into the problem.
- Perform "thought experiments" to explore the effects of different conditions for theoretical problems. When troubleshooting real systems, perform *diagnostic tests* rather than visually inspecting for faults, the best diagnostic test being the one giving you the most information about the nature and/or location of the fault with the fewest steps.
- Simplify the problem until the solution becomes obvious, and then use that obvious case as a model to follow in solving the more complex version of the problem.
- Check for exceptions to see if your solution is incorrect or incomplete. A good solution will work for *all* known conditions and criteria. A good example of this is the process of testing scientific hypotheses: the task of a scientist is not to find support for a new idea, but rather to *challenge* that new idea to see if it holds up under a battery of tests. The philosophical

principle of *reductio ad absurdum* (i.e. disproving a general idea by finding a specific case where it fails) is useful here.

- Work "backward" from a hypothetical solution to a new set of given conditions.
- Add quantities to problems that are qualitative in nature, because sometimes a little math helps illuminate the scenario.
- Sketch graphs illustrating how variables relate to each other. These may be quantitative (i.e. with realistic number values) or qualitative (i.e. simply showing increases and decreases).
- Treat quantitative problems as qualitative in order to discern the relative magnitudes and/or directions of change of the relevant variables. For example, try determining what happens if a certain variable were to increase or decrease before attempting to precisely calculate quantities: how will each of the dependent variables respond, by increasing, decreasing, or remaining the same as before?
- Consider limiting cases. This works especially well for qualitative problems where you need to determine which direction a variable will change. Take the given condition and magnify that condition to an extreme degree as a way of simplifying the direction of the system's response.
- Check your work. This means regularly testing your conclusions to see if they make sense. This does *not* mean repeating the same steps originally used to obtain the conclusion(s), but rather to use some other means to check validity. Simply repeating procedures often leads to *repeating the same errors* if any were made, which is why alternative paths are better.

Appendix B

Instructional philosophy

"The unexamined circuit is not worth energizing" – Socrates (if he had taught electricity)

These learning modules, although useful for self-study, were designed to be used in a formal learning environment where a subject-matter expert challenges students to digest the content and exercise their critical thinking abilities in the answering of questions and in the construction and testing of working circuits.

The following principles inform the instructional and assessment philosophies embodied in these learning modules:

- The first goal of education is to enhance clear and independent thought, in order that every student reach their fullest potential in a highly complex and inter-dependent world. Robust reasoning is *always* more important than particulars of any subject matter, because its application is universal.
- Literacy is fundamental to independent learning and thought because text continues to be the most efficient way to communicate complex ideas over space and time. Those who cannot read with ease are limited in their ability to acquire knowledge and perspective.
- Articulate communication is fundamental to work that is complex and interdisciplinary.
- Faulty assumptions and poor reasoning are best corrected through challenge, not presentation. The rhetorical technique of *reductio ad absurdum* (disproving an assertion by exposing an absurdity) works well to discipline student's minds, not only to correct the problem at hand but also to learn how to detect and correct future errors.
- Important principles should be repeatedly explored and widely applied throughout a course of study, not only to reinforce their importance and help ensure their mastery, but also to showcase the interconnectedness and utility of knowledge.

These learning modules were expressly designed to be used in an "inverted" teaching environment[1] where students first read the introductory and tutorial chapters on their own, then individually attempt to answer the questions and construct working circuits according to the experiment and project guidelines. The instructor never lectures, but instead meets regularly with each individual student to review their progress, answer questions, identify misconceptions, and challenge the student to new depths of understanding through further questioning. Regular meetings between instructor and student should resemble a Socratic[2] dialogue, where questions serve as scalpels to dissect topics and expose assumptions. The student passes each module only after consistently demonstrating their ability to logically analyze and correctly apply all major concepts in each question or project/experiment. The instructor must be vigilant in probing each student's understanding to ensure they are truly *reasoning* and not just *memorizing*. This is why "Challenge" points appear throughout, as prompts for students to think deeper about topics and as starting points for instructor queries. Sometimes these challenge points require additional knowledge that hasn't been covered in the series to answer in full. This is okay, as the major purpose of the Challenges is to stimulate analysis and synthesis on the part of each student.

The instructor must possess enough mastery of the subject matter and awareness of students' reasoning to generate their own follow-up questions to practically any student response. Even completely correct answers given by the student should be challenged by the instructor for the purpose of having students practice articulating their thoughts and defending their reasoning. Conceptual errors committed by the student should be exposed and corrected not by direct instruction, but rather by reducing the errors to an absurdity[3] through well-chosen questions and thought experiments posed by the instructor. Becoming proficient at this style of instruction requires time and dedication, but the positive effects on critical thinking for both student and instructor are spectacular.

An inspection of these learning modules reveals certain unique characteristics. One of these is a bias toward thorough explanations in the tutorial chapters. Without a live instructor to explain concepts and applications to students, the text itself must fulfill this role. This philosophy results in lengthier explanations than what you might typically find in a textbook, each step of the reasoning process fully explained, including footnotes addressing common questions and concerns students raise while learning these concepts. Each tutorial seeks to not only explain each major concept in sufficient detail, but also to explain the logic of each concept and how each may be developed

[1] In a traditional teaching environment, students first encounter new information via *lecture* from an expert, and then independently apply that information via *homework*. In an "inverted" course of study, students first encounter new information via *homework*, and then independently apply that information under the scrutiny of an expert. The expert's role in lecture is to simply *explain*, but the expert's role in an inverted session is to *challenge*, *critique*, and if necessary *explain* where gaps in understanding still exist.

[2] Socrates is a figure in ancient Greek philosophy famous for his unflinching style of questioning. Although he authored no texts, he appears as a character in Plato's many writings. The essence of Socratic philosophy is to leave no question unexamined and no point of view unchallenged. While purists may argue a topic such as electric circuits is too narrow for a true Socratic-style dialogue, I would argue that the essential thought processes involved with scientific reasoning on *any* topic are not far removed from the Socratic ideal, and that students of electricity and electronics would do very well to challenge assumptions, pose thought experiments, identify fallacies, and otherwise employ the arsenal of critical thinking skills modeled by Socrates.

[3] This rhetorical technique is known by the Latin phrase *reductio ad absurdum*. The concept is to expose errors by counter-example, since only one solid counter-example is necessary to disprove a universal claim. As an example of this, consider the common misconception among beginning students of electricity that voltage cannot exist without current. One way to apply *reductio ad absurdum* to this statement is to ask how much current passes through a fully-charged battery connected to nothing (i.e. a clear example of voltage existing without current).

from "first principles". Again, this reflects the goal of developing clear and independent thought in students' minds, by showing how clear and logical thought was used to forge each concept. Students benefit from witnessing a model of clear thinking in action, and these tutorials strive to be just that.

Another characteristic of these learning modules is a lack of step-by-step instructions in the Project and Experiment chapters. Unlike many modern workbooks and laboratory guides where step-by-step instructions are prescribed for each experiment, these modules take the approach that students must learn to closely read the tutorials and apply their own reasoning to identify the appropriate experimental steps. Sometimes these steps are plainly declared in the text, just not as a set of enumerated points. At other times certain steps are implied, an example being assumed competence in test equipment use where the student should not need to be told *again* how to use their multimeter because that was thoroughly explained in previous lessons. In some circumstances no steps are given at all, leaving the entire procedure up to the student.

This lack of prescription is not a flaw, but rather a feature. Close reading and clear thinking are foundational principles of this learning series, and in keeping with this philosophy all activities are designed to *require* those behaviors. Some students may find the lack of prescription frustrating, because it demands more from them than what their previous educational experiences required. This frustration should be interpreted as an unfamiliarity with autonomous thinking, a problem which must be corrected if the student is ever to become a self-directed learner and effective problem-solver. Ultimately, the need for students to read closely and think clearly is more important both in the near-term and far-term than any specific facet of the subject matter at hand. If a student takes longer than expected to complete a module because they are forced to outline, digest, and reason on their own, so be it. The future gains enjoyed by developing this mental discipline will be well worth the additional effort and delay.

Another feature of these learning modules is that they do not treat topics in isolation. Rather, important concepts are introduced early in the series, and appear repeatedly as stepping-stones toward other concepts in subsequent modules. This helps to avoid the "compartmentalization" of knowledge, demonstrating the inter-connectedness of concepts and simultaneously reinforcing them. Each module is fairly complete in itself, reserving the beginning of its tutorial to a review of foundational concepts.

This methodology of assigning text-based modules to students for digestion and then using Socratic dialogue to assess progress and hone students' thinking was developed over a period of several years by the author with his Electronics and Instrumentation students at the two-year college level. While decidedly unconventional and sometimes even unsettling for students accustomed to a more passive lecture environment, this instructional philosophy has proven its ability to convey conceptual mastery, foster careful analysis, and enhance employability so much better than lecture that the author refuses to ever teach by lecture again.

Problems which often go undiagnosed in a lecture environment are laid bare in this "inverted" format where students must articulate and logically defend their reasoning. This, too, may be unsettling for students accustomed to lecture sessions where the instructor cannot tell for sure who comprehends and who does not, and this vulnerability necessitates sensitivity on the part of the "inverted" session instructor in order that students never feel discouraged by having their errors exposed. *Everyone* makes mistakes from time to time, and learning is a lifelong process! Part of the instructor's job is to build a culture of learning among the students where errors are not seen as shameful, but rather as opportunities for progress.

To this end, instructors managing courses based on these modules should adhere to the following principles:

- Student questions are always welcome and demand thorough, honest answers. The only type of question an instructor should refuse to answer is one the student should be able to easily answer on their own. Remember, *the fundamental goal of education is for each student to learn to think clearly and independently.* This requires hard work on the part of the student, which no instructor should ever circumvent. Anything done to bypass the student's responsibility to do that hard work ultimately limits that student's potential and thereby does real harm.

- It is not only permissible, but encouraged, to answer a student's question by asking questions in return, these follow-up questions designed to guide the student to reach a correct answer through their own reasoning.

- All student answers demand to be challenged by the instructor and/or by other students. This includes both correct and incorrect answers – the goal is to practice the articulation and defense of one's own reasoning.

- No reading assignment is deemed complete unless and until the student demonstrates their ability to accurately summarize the major points in their own terms. Recitation of the original text is unacceptable. This is why every module contains an "Outline and reflections" question as well as a "Foundational concepts" question in the Conceptual reasoning section, to prompt reflective reading.

- No assigned question is deemed answered unless and until the student demonstrates their ability to consistently and correctly apply the concepts to *variations* of that question. This is why module questions typically contain multiple "Challenges" suggesting different applications of the concept(s) as well as variations on the same theme(s). Instructors are encouraged to devise as many of their own "Challenges" as they are able, in order to have a multitude of ways ready to probe students' understanding.

- No assigned experiment or project is deemed complete unless and until the student demonstrates the task in action. If this cannot be done "live" before the instructor, video-recordings showing the demonstration are acceptable. All relevant safety precautions must be followed, all test equipment must be used correctly, and the student must be able to properly explain all results. The student must also successfully answer all Challenges presented by the instructor for that experiment or project.

Students learning from these modules would do well to abide by the following principles:

- No text should be considered fully and adequately read unless and until you can express every idea *in your own words, using your own examples.*
- You should always articulate your thoughts as you read the text, noting points of agreement, confusion, and epiphanies. Feel free to print the text on paper and then write your notes in the margins. Alternatively, keep a journal for your own reflections as you read. This is truly a helpful tool when digesting complicated concepts.
- Never take the easy path of highlighting or underlining important text. Instead, *summarize* and/or *comment* on the text using your own words. This actively engages your mind, allowing you to more clearly perceive points of confusion or misunderstanding on your own.
- A very helpful strategy when learning new concepts is to place yourself in the role of a teacher, if only as a mental exercise. Either explain what you have recently learned to someone else, or at least *imagine* yourself explaining what you have learned to someone else. The simple act of having to articulate new knowledge and skill forces you to take on a different perspective, and will help reveal weaknesses in your understanding.
- Perform each and every mathematical calculation and thought experiment shown in the text on your own, referring back to the text to see that your results agree. This may seem trivial and unnecessary, but it is critically important to ensuring you actually understand what is presented, especially when the concepts at hand are complicated and easy to misunderstand. Apply this same strategy to become proficient in the use of *circuit simulation software*, checking to see if your simulated results agree with the results shown in the text.
- Above all, recognize that learning is hard work, and that a certain level of frustration is unavoidable. There are times when you will struggle to grasp some of these concepts, and that struggle is a natural thing. Take heart that it will yield with persistent and varied[4] effort, and never give up!

Students interested in using these modules for self-study will also find them beneficial, although the onus of responsibility for thoroughly reading and answering questions will of course lie with that individual alone. If a qualified instructor is not available to challenge students, a workable alternative is for students to form study groups where they challenge[5] one another.

To high standards of education,

Tony R. Kuphaldt

---

[4] As the old saying goes, "Insanity is trying the same thing over and over again, expecting different results." If you find yourself stumped by something in the text, you should attempt a different approach. Alter the thought experiment, change the mathematical parameters, do whatever you can to see the problem in a slightly different light, and then the solution will often present itself more readily.

[5] Avoid the temptation to simply share answers with study partners, as this is really counter-productive to learning. Always bear in mind that the answer to any question is far less important in the long run than the method(s) used to obtain that answer. The goal of education is to empower one's life through the improvement of clear and independent thought, literacy, expression, and various practical skills.

Appendix C

Tools used

I am indebted to the developers of many open-source software applications in the creation of these learning modules. The following is a list of these applications with some commentary on each.

You will notice a theme common to many of these applications: a bias toward *code*. Although I am by no means an expert programmer in any computer language, I understand and appreciate the flexibility offered by code-based applications where the user (you) enters commands into a plain ASCII text file, which the software then reads and processes to create the final output. Code-based computer applications are by their very nature *extensible*, while WYSIWYG (What You See Is What You Get) applications are generally limited to whatever user interface the developer makes for you.

The `GNU/Linux` computer operating system

> There is so much to be said about Linus Torvalds' `Linux` and Richard Stallman's `GNU` project. First, to credit just these two individuals is to fail to do justice to the *mob* of passionate volunteers who contributed to make this amazing software a reality. I first learned of `Linux` back in 1996, and have been using this operating system on my personal computers almost exclusively since then. It is *free*, it is completely *configurable*, and it permits the continued use of highly efficient `Unix` applications and scripting languages (e.g. shell scripts, Makefiles, `sed`, `awk`) developed over many decades. `Linux` not only provided me with a powerful computing platform, but its open design served to inspire my life's work of creating open-source educational resources.

Bram Moolenaar's `Vim` text editor

> Writing code for any code-based computer application requires a *text editor*, which may be thought of as a word processor strictly limited to outputting plain-ASCII text files. Many good text editors exist, and one's choice of text editor seems to be a deeply personal matter within the programming world. I prefer `Vim` because it operates very similarly to `vi` which is ubiquitous on `Unix`/`Linux` operating systems, and because it may be entirely operated via keyboard (i.e. no mouse required) which makes it fast to use.

Donald Knuth's TEX typesetting system

Developed in the late 1970's and early 1980's by computer scientist extraordinaire Donald Knuth to typeset his multi-volume magnum opus *The Art of Computer Programming*, this software allows the production of formatted text for screen-viewing or paper printing, all by writing plain-text code to describe how the formatted text is supposed to appear. TEX is not just a markup language for documents, but it is also a Turing-complete programming language in and of itself, allowing useful algorithms to be created to control the production of documents. Simply put, TEX *is a programmer's approach to word processing*. Since TEX is controlled by code written in a plain-text file, this means anyone may read that plain-text file to see exactly how the document was created. This openness afforded by the code-based nature of TEX makes it relatively easy to learn how other people have created their own TEX documents. By contrast, examining a beautiful document created in a conventional WYSIWYG word processor such as Microsoft `Word` suggests nothing to the reader about *how* that document was created, or what the user might do to create something similar. As Mr. Knuth himself once quipped, conventional word processing applications should be called WYSIAYG (What You See Is *All* You Get).

Leslie Lamport's LATEX extensions to TEX

Like all true programming languages, TEX is inherently extensible. So, years after the release of TEX to the public, Leslie Lamport decided to create a massive extension allowing easier compilation of book-length documents. The result was LATEX, which is the markup language used to create all ModEL module documents. You could say that TEX is to LATEX as `C` is to `C++`. This means it is permissible to use any and all TEX commands within LATEX source code, and it all still works. Some of the features offered by LATEX that would be challenging to implement in TEX include automatic index and table-of-content creation.

Tim Edwards' `Xcircuit` drafting program

This wonderful program is what I use to create all the schematic diagrams and illustrations (but not photographic images or mathematical plots) throughout the ModEL project. It natively outputs PostScript format which is a true vector graphic format (this is why the images do not pixellate when you zoom in for a closer view), and it is so simple to use that I have never had to read the manual! Object libraries are easy to create for `Xcircuit`, being plain-text files using PostScript programming conventions. Over the years I have collected a large set of object libraries useful for drawing electrical and electronic schematics, pictorial diagrams, and other technical illustrations.

`Gimp` graphic image manipulation program

Essentially an open-source clone of Adobe's `PhotoShop`, I use `Gimp` to resize, crop, and convert file formats for all of the photographic images appearing in the ModEL modules. Although `Gimp` does offer its own scripting language (called `Script-Fu`), I have never had occasion to use it. Thus, my utilization of `Gimp` to merely crop, resize, and convert graphic images is akin to using a sword to slice bread.

`SPICE` circuit simulation program

`SPICE` is to circuit analysis as TEX is to document creation: it is a form of markup language designed to describe a certain object to be processed in plain-ASCII text. When the plain-text "source file" is compiled by the software, it outputs the final result. More modern circuit analysis tools certainly exist, but I prefer `SPICE` for the following reasons: it is *free*, it is *fast*, it is *reliable*, and it is a fantastic tool for *teaching* students of electricity and electronics how to write simple code. I happen to use rather old versions of `SPICE`, version 2g6 being my "go to" application when I only require text-based output. `NGSPICE` (version 26), which is based on Berkeley `SPICE` version 3f5, is used when I require graphical output for such things as time-domain waveforms and Bode plots. In all `SPICE` example netlists I strive to use coding conventions compatible with all `SPICE` versions.

Andrew D. Hwang's `ePiX` mathematical visualization programming library

This amazing project is a `C++` library you may link to any `C/C++` code for the purpose of generating PostScript graphic images of mathematical functions. As a completely free and open-source project, it does all the plotting I would otherwise use a Computer Algebra System (CAS) such as `Mathematica` or `Maple` to do. It should be said that `ePiX` is *not* a Computer Algebra System like `Mathematica` or `Maple`, but merely a mathematical *visualization* tool. In other words, it won't determine integrals for you (you'll have to implement that in your own `C/C++` code!), but it can graph the results, and it does so beautifully. What I really admire about `ePiX` is that it is a `C++` programming library, which means it builds on the existing power and toolset available with that programming language. Mr. Hwang could have probably developed his own stand-alone application for mathematical plotting, but by creating a `C++` library to do the same thing he accomplished something much greater.

`gnuplot` mathematical visualization software

Another open-source tool for mathematical visualization is `gnuplot`. Interestingly, this tool is *not* part of Richard Stallman's GNU project, its name being a coincidence. For this reason the authors prefer "gnu" *not* be capitalized at all to avoid confusion. This is a much "lighter-weight" alternative to a spreadsheet for plotting tabular data, and the fact that it easily outputs directly to an X11 console or a file in a number of different graphical formats (including PostScript) is very helpful. I typically set my `gnuplot` output format to default (X11 on my Linux PC) for quick viewing while I'm developing a visualization, then switch to PostScript file export once the visual is ready to include in the document(s) I'm writing. As with my use of `Gimp` to do rudimentary image editing, my use of `gnuplot` only scratches the surface of its capabilities, but the important points are that it's *free* and that it *works well*.

`Python` programming language

Both Python and C++ find extensive use in these modules as instructional aids and exercises, but I'm listing Python here as a *tool* for myself because I use it almost daily as a *calculator*. If you open a Python interpreter console and type `from math import *` you can type mathematical expressions and have it return results just as you would on a hand calculator. Complex-number (i.e. *phasor*) arithmetic is similarly supported if you include the complex-math library (`from cmath import *`). Examples of this are shown in the Programming References chapter (if included) in each module. Of course, being a fully-featured programming language, Python also supports conditionals, loops, and other structures useful for calculation of quantities. Also, running in a console environment where all entries and returned values show as text in a chronologically-ordered list makes it easy to copy-and-paste those calculations to document exactly how they were performed.

Appendix D

Creative Commons License

Creative Commons Attribution 4.0 International Public License

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution 4.0 International Public License ("Public License"). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

Section 1 – Definitions.

a. **Adapted Material** means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.

b. **Adapter's License** means the license You apply to Your Copyright and Similar Rights in Your contributions to Adapted Material in accordance with the terms and conditions of this Public License.

c. **Copyright and Similar Rights** means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights.

d. **Effective Technological Measures** means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.

e. **Exceptions and Limitations** means fair use, fair dealing, and/or any other exception or

limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.

f. **Licensed Material** means the artistic or literary work, database, or other material to which the Licensor applied this Public License.

g. **Licensed Rights** means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.

h. **Licensor** means the individual(s) or entity(ies) granting rights under this Public License.

i. **Share** means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.

j. **Sui Generis Database Rights** means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.

k. **You** means the individual or entity exercising the Licensed Rights under this Public License. **Your** has a corresponding meaning.

Section 2 – Scope.

a. License grant.

1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:

A. reproduce and Share the Licensed Material, in whole or in part; and

B. produce, reproduce, and Share Adapted Material.

2. Exceptions and Limitations. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.

3. Term. The term of this Public License is specified in Section 6(a).

4. Media and formats; technical modifications allowed. The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures.

For purposes of this Public License, simply making modifications authorized by this Section 2(a)(4) never produces Adapted Material.

5. Downstream recipients.

A. Offer from the Licensor – Licensed Material. Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.

B. No downstream restrictions. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.

6. No endorsement. Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).

b. Other rights.

1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.

2. Patent and trademark rights are not licensed under this Public License.

3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties.

Section 3 – License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. Attribution.

1. If You Share the Licensed Material (including in modified form), You must:

A. retain the following if it is supplied by the Licensor with the Licensed Material:

i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);

ii. a copyright notice;

iii. a notice that refers to this Public License;

iv. a notice that refers to the disclaimer of warranties;

v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;

B. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and

C. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.

2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.

3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.

4. If You Share Adapted Material You produce, the Adapter's License You apply must not prevent recipients of the Adapted Material from complying with this Public License.

Section 4 – Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database;

b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material; and

c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

Section 5 – Disclaimer of Warranties and Limitation of Liability.

a. Unless otherwise separately undertaken by the Licensor, to the extent possible, the Licensor offers the Licensed Material as-is and as-available, and makes no representations or warranties of any kind concerning the Licensed Material, whether express, implied, statutory, or other. This includes, without limitation, warranties of title, merchantability, fitness for a particular purpose, non-infringement, absence of latent or other defects, accuracy, or the presence or absence of errors,

whether or not known or discoverable. Where disclaimers of warranties are not allowed in full or in part, this disclaimer may not apply to You.

b. To the extent possible, in no event will the Licensor be liable to You on any legal theory (including, without limitation, negligence) or otherwise for any direct, special, indirect, incidental, consequential, punitive, exemplary, or other losses, costs, expenses, or damages arising out of this Public License or use of the Licensed Material, even if the Licensor has been advised of the possibility of such losses, costs, expenses, or damages. Where a limitation of liability is not allowed in full or in part, this limitation may not apply to You.

c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

Section 6 – Term and Termination.

a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.

b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:

1. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or

2. upon express reinstatement by the Licensor.

For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

c. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.

d. Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

Section 7 – Other Terms and Conditions.

a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.

b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

Section 8 – Interpretation.

a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully

be made without permission under this Public License.

b. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.

c. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.

d. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.

Creative Commons is not a party to its public licenses. Notwithstanding, Creative Commons may elect to apply one of its public licenses to material it publishes and in those instances will be considered the "Licensor." Except for the limited purpose of indicating that material is shared under a Creative Commons public license or as otherwise permitted by the Creative Commons policies published at `creativecommons.org/policies`, Creative Commons does not authorize the use of the trademark "Creative Commons" or any other trademark or logo of Creative Commons without its prior written consent including, without limitation, in connection with any unauthorized modifications to any of its public licenses or any other arrangements, understandings, or agreements concerning use of licensed material. For the avoidance of doubt, this paragraph does not form part of the public licenses.

Creative Commons may be contacted at `creativecommons.org`.

Appendix E

References

CODATA Recommended Values of the Fundamental Physical Constants, NIST SP 961, National Institute of Standards and Technology, May 2019.

Denker, John S., "Uncertainty as Applied to Measurements and Calculations".

NIST/SEMATECH e-Handbook of Statistical Methods, `http://www.itl.nist.gov/div898/handbook`, National Institute of Standards and Technology, sampled 9 June 2022.

Taylor, Barry N. and Kuyatt, Chris E., "Guidelines for Evaluating and Expressing the Uncertainty of NIST Measurement Results", NIST Technical Note 1297, 1994 Edition, National Institute of Standards and Technology, September 1994. "

Appendix F

Version history

This is a list showing all significant additions, corrections, and other edits made to this learning module. Each entry is referenced by calendar date in reverse chronological order (newest version first), which appears on the front cover of every learning module for easy reference. Any contributors to this open-source document are listed here as well.

2-3 February 2024 – added more discussion to the "Significant figures" section of the Tutorial, and also added comments regarding asymmetrical probability distributions.

22 June 2023 – added some comments about "significant figures" rules and how they confuse the real concept of uncertainty.

7 June 2023 – corrected a quantitative error where I showed the power-of-ten for the metric prefix Exa as 15 rather than 18 as it should have been.

30 August 2022 – corrected a quantitative error regarding the expression of uncertainty for the mass of a proton, and also elaborated slightly on the practical example of a Gaussian distribution.

24 June 2022 – added questions to the module.

9 June 2022 – minor additions to the Tutorial chapter, mostly dealing with ppm and ppb measurements. Also added a Tutorial section on measurement uncertainty.

3 June 2021 – moved "Resistor labeling" content to its own included file, and included with it letter-based tolerance codes for resistance.

18-19 May 2021 – document first created.

Index