Variables

We already used `pi` above from the `math` module as a variable. Variables are easily initiated in Python, since Python is a dynamically typed language, meaning you don't have to tell it what kind of variable you want to save in memory. It will deduce that from the value you're assigning to the variable:

In [4]:

```
import math as m

a = 78.2
i = 15
s = "characters"
f = m.sin

print(type(a))
print(type(i))
print(type(s))
print(type(f))
```

```
<class 'float'>
<class 'int'>
<class 'str'>
<class 'builtin_function_or_method'>
```

This does come with detriments. E.g., you can never be sure what type a variable gets after assignment, so you should make sure that if you need a floating point variable, that you initiate it with a floating point number. This is just a safety measure, usually Python does the thinking here for the programmer. Speaking of floating point variables, perhaps you noticed the sine of π wasn't exactly zero in the example from last lesson. This is caused by machine precision, or the finite representation of numbers in computers. Some languages do not tell the truth regarding this limitation. See for example what Excel/LibreOffice Calc or similar tell you, when you ask `=IF(0,1 + 0,1 + 0,1 = 0,3;"TRUE";"FALSE")` . Compare this to the answer that Python gives:

In [7]:

```
print(0.1 + 0.1 + 0.1 == 0.3)
```

```
False
```

This causes some surprising effects, for example cancellation, where subtracting numbers close in value might lead to significant loss of precision. The following example is taken from https://math.stackexchange.com/questions/1920525/why-is-catastrophic-cancellation-called-so (https://math.stackexchange.com/questions/1920525/why-is-catastrophic-cancellation-called-so). Take for example the polynomial $x^2 - 1000.0x + 1.0 = 0.0$. The midnight formula gives $x_{1/2} = \frac{1000.0 \pm 999.99}{2}$

In [4]:

```
# roots:
x1 = (1000 + 999.99)/2
x2 = (1000 - 999.99)/2

print("Analytical roots:\t", x1, "and", x2)

# While the first is almost correct, the second one isn't at all
x1c = 1000
x2c = 0.0010005

print("x1 error:\t\t", 100*(x1 - x1c)/x1c, "%")
print("x2 error:\t\t", 100*(x2 - x2c)/x2c, "%")
```

```
Analytical roots:	 999.995 and 0.0049999999999954525
x1 error:		 -0.0004999999999995453 %
x2 error:		 399.7501249370767 %
```

For more information regarding floating point arithmetic and its limitations and implications, see this exhaustive document: https://docs.oracle.com/cd/E19422-01/819-3693/ncg_goldberg.html (https://docs.oracle.com/cd/E19422-01/819-3693/ncg_goldberg.html)

The `==` here is a comparison operator. These work quite intuitively. Same applies to gate operators (`^` is `xor`):

In [5]:

```
a = 14
b = True

print(11 < 13)
print(11 > 13)
print(11 == 13)
print(11 != a)
print(b)
print(not b)
print(11 < 13 and b)
print(11 < 13 and not b)
print(b or not b)
print((not b) ^ (not b))
```

```
True
False
False
True
True
False
True
False
True
False
```