In [2]:
```
# Styling notebook
from IPython.core.display import HTML
def css_styling():
    styles = open("./styles/custom.css", "r").read()
    return HTML(styles)
css_styling()
```

Out[2]:

## Number systems

Idea: Use a different base from 10 to write down numbers (usually non-negative integers)

Common bases: 2, 8, 16

**Important:** Given a base $b > 1$, the "digits" of this base system are the remainders mod $b$.

Digits < 10 are usually written like in the decimal system $0, 1, \dots, 9$.

For 10 and greater we use characters a, b, c, ... (here generally lowercase)

In [3]:
```
# return char for "digit" in "base" ("digit" is a remainder mod "base")
#    if digit <= 9, it's just the digit as a char
#    else 10 --> 'a', 11 -- 'b', etc
def chr4Digit(digit, base) :
    assert (digit >= 0 and digit < base),"Not a legal digit!"

    # This looks SOOO much like Pascal!
    if digit <= 9 :
        return chr(ord("0") + digit)
    else :
        return chr(ord("a") + digit - 10)

# the inverse of "chr4Digit"
#    '0' - '9' --> 0 - 9
#    'a' --> 10, 'b' --> 11 etc
def digit4Chr(c) :
    if (c >= '0' and c <= '9') :
        return ord(c) - ord('0')
    else :
        return ord(c) - ord('a') + 10

# -------------------------------- Just to demo the digits to base b -----------

def allDigits(b) : # return a list of all digits for a given base b
    result = []
    for r in range(b) :
        result.append(chr4Digit(r,b))
    return result

allDigits(13)
```

Out[3]: ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c']

**Value for arbitrary bases**

Given a base $b > 1$, and the digits $0, 1, \ldots, b-2, b-1,$ a number to base $b$ is a sequence of base b digits: $d_n d_{n-1} d_{n-2} \ldots d_2 d_1 d_0$.

The value of this number is then given by the polynomial

$$\sum_{i=0}^{n} d_i b^i$$

Since the indices are assumed to be decreasing, the above formula can be written out as

$$d_n b^n + d_{n-1} b^{n-1} + d_{n-2} b^{n-2} + \ldots + d_2 b^2 + d_1 b + d_0$$

**Base conversions**

Three directions:

1. Convert a string (!) with a base $b$ encoding into an integer
2. Convert an integer to a base $b$ string
3. Convert a string in base $b_1$ encoding into a string in $b_2$ encoding

No demos for switching bases 2, 4, 8, 16 etc. Why? It doesn't make sense! These grouping methods are strictly for quick manual conversions - if you have access to converter software (which you do now!) just use that!

In [4]:
```python
# Converts a string "s" containing the encoding of an int in base "b" into the va
# Just evaluate the polynomial = best from right to left, to have the exponents i

def fromBase(s, b) :
    result = 0
    factor = 1

    for i in range(len(s)-1, -1, -1) :  # cute way of running backwards through a
        result += digit4Chr(s[i])*factor
        factor *= b

    return result
```

In [5]:
```python
fromBase('cafe',16)
```

Out[5]:
```
51966
```

**Converting from base 10 to any other base b**

Repeatedly divide by b until quotient is 0

Write remainders (as b-digits) right to left (use extra “digits” if necessary)

In [6]:
```
# convert int "n" to a string for base "b"
def toBase(n, b) :
    result = ''

    while n >= b :
        result = chr4Digit(n%b,b) + result # <-- put the next character in front!
        n = n//b

    return chr4Digit(n,b) + result

toBase(510,19)
```

Out[6]: '17g'

In [7]:
```
# convert string "s" (assuming it's in base "b1")
# into the string for base "b2"
def changeBase(s,b1,b2) :
    return toBase(fromBase(s,b1),b2)
```

In [10]:
```
changeBase('145376',8,2)
```

Out[10]: '1100101011111110'