**Problem #1**

Subject: Formatting a large integer with arbitrary separators

Filename: pretty_number.py
Allowed modules: NONE

**WORK**

To achieve this, I have to start from scratch because I my original pretty_int() code does not easily support changing the block size. To correct this program, I need to convert n to a string, and add “sep” in between each block of numbers. I can do this by stepping backwards through the string with an increment of “-group”, and then add “sep” until I reach the beginning.

**OUTPUT**

```
-------Pretty Int Test--------

0 = 0

1 = 1

999 = 999

1000 = 1,000

65536 = 65,536

18446744073709551616 = 18,446,744,073,709,551,616

1000 = 1000

65536 = 6-5536

18446744073709551616 = 1844-6744-0737-0955-1616
```

**CODE**

```
def pretty_int(n, sep = ',', group = 3):

  num_str = str(n)

  #handle group size of 0, return original number as a string

  if group <= 0:

    return str(n)
```

```
result = ''
#step backwards through number by increment of -group
for i in range(len(num_str), 0, -group):
    #if there are less numbers than group size, add numbers to beginning of string
    if i - group <= 0:
        result = num_str[:i] + result
    #otherwise, set 'result' string to seperator and remaining numbers in result
    else:
        result = sep + num_str[i - group:i] + result
return result
```

**Problem #2**

Subject: Formatting a floating point value with arbitrary separators.

Filename: pretty_number.py
Allowed modules:

**WORK**

To get this function to work I just need to handle positive inputs, negative inputs, and floating numbers. I will return pretty_int(n, sep, group) if the number is positive or 0, '-' + pretty_int() of the absolute value of n if the number is negative, and pretty_int() of the value before the decimal + the remaining portion of the number if it is a float.

**OUTPUT**

```
-------Pretty Num Test--------

0 = 0

999 = 999

1000 = 1,000

0 = 0

-1 = -1

-999 = -999

-1000 = -1,000

-65536 = -65,536

-18446744073709551616 = -18,446,744,073,709,551,616

0.1234 = 0.123

1000.0 = 1,000.0

65536.0625 = 65,536.062

-65535.9375 = -65,535.938
```

**CODE**

```
def pretty_num(n, sep = ',', group = 3, places = 6, mark = '.'):

  #determine whether number is a float or not
```

```
isFloat = False
if '.' in str(n):
   isFloat = True

#number is not a float
if isFloat == False:
   #if number is positive
   if n >= 0:
      return pretty_int(n,sep, group)
   #if number is negative
   else:
      return '-' +  pretty_int(abs(n),sep, group)

#number is a float
else:
   n = round(n, places)
   num_float = pretty_num(int(str(n).split('.')[0]), sep, group) + mark + str(n).split('.')[1]
   return num_float
```

**<u>Problem #3</u>**

Subject: Formatting a value significant figures.

Filename: pretty_number.py
Allowed modules:

**<u>WORK</u>**

For this problem, I will just need to round the number to the correct sig figs. If the length of the number is greater than sigfigs, I will round to (sigfigs – length), otherwise I will round to sigfigs and add trailing 0s until the number reaches sigfigs.

**<u>OUTPUT</u>**

```
 --------Pretty SF Test--------

0 = 0.00000

3.14159276535 = 3.1416

1 = 1.0000

999 = 999.00

1000 = 1000.0

65536 = 65536

18446744073709551616 = 18447000000000000000
```

**<u>CODE</u>**

```
 def pretty_sf(n, sigfigs = 3):

    str_num = ''

    #handle negative

    if n < 0:

        str_num += '-'

        n = abs(n)

    #handle 0

    if n == 0:

        return '0.' + '0' * (sigfigs)
```

```
#remove leading 0s
str_n = str(n).lstrip('0')

#handle decimal values
if '.' in str(n):
  length = len(str(n).split('.')[0]) + len(str(n).split('.')[1])
  str_num += str(round(n,sigfigs-1))
#handle every other value
else:
  length = len(str(n))
  if length < sigfigs:
    str_num += str(round(n,sigfigs))
    str_num += '.'
  else:
    str_num += str(round(n,sigfigs-length))
while (length) < sigfigs:
  str_num += '0'
  length += 1

return str_num
```

**Problem #4**

Subject: Formatting a value with significant figures.

Filename: pretty_number.py
Allowed modules:

**WORK**

I will split this problem into 4 separate parts. To handle negatives, I will treat it as a positive number and add a '-' to the return string. To handle 0s, I will return '0.' + 0 * sigfigs. Now the remaining parts to handle will be cases where the number is a decimal less than 1, and where the number is above 1000. To handle the decimal number, I will multiply by 1000 until the value is above or equal to 1, and adjust in smaller steps of 10 to make sure the power is a multiple of 3. To handle the last case, I will do the same process as the decimal case, but with dividing by 1000.

**OUTPUT**

```
--------Pretty SI Test--------

0 = 0.00

0.03125 = 31.25m

1 = 1

999 = 999

1000 = 1.0k

65536 = 65.54k

-65536 = -65.54k

18446744073709551616 = 18.45E

340282366920938463463374607431768211456 = 340.28e36
```

**CODE**

```
def pretty_si(n, si=False, sigfigs = 3):


  prefix_table = {30: "Q", 27: "R",

  24: "Y", 21: "Z", 18: "E",

  15: "P", 12: "T", 9: "G",
  6: "M", 3: "k", -3: "m",
  -6: "µ", -9: "n", -12: "p",
  -15: "f", -18: "a", -21: "z",
  -24: "y", -27: "r", -30: "q"
}

  power = 0
  str_num = ''

  if n < -1:
      str_num += '-'
      n = abs(n)

  if n == 0:
      return '0.' + '0' * (sigfigs - 1)

  if n >= 1:
      while n>=1000:
          n /= 1000
          power += 3
      while (power%3 != 0):
          n /= 10
          power += 1
  #handle decimals between -1 and 1
  else:
      while n<1:
          n *= 1000
          power -= 3
    while (abs(power)%3 != 0):
      n *= 10
      power -= 1

  if power != 0:
    if power in range (-30,30):
      str_num += str(round(n, sigfigs - 1)) + prefix_table[power]
    else:
      str_num += str(round(n, sigfigs - 1)) + 'e' + str(power)
  else:
    str_num += str(round(n, sigfigs - 1))

  return str_num
```