```python
def decimal_to_floating_point(x, n, p):
    """
    x: float
        Input positive decimal number
    n: int
        Number of binary bits in fraction
    p: int
        Exponent range
    """

    # your implementation goes here

    integer_part = int(x)
    decimal_part = x - integer_part
    int_bin = bin(integer_part)[2:]

    def dec2bin(x):
        x -= int(x)
        bins = []

        while x:
            x *= 2
            bins.append(1 if x>=1. else 0)
            x -= int(x)

        return bins
    decimal_bins = dec2bin(decimal_part)
    if integer_part != 0:
        m = len(str(int_bin)) - 1
        f = str(int_bin)[1:]
        for i in decimal_bins:
            f += str(i)
        if len(f) > n:
            f = f[:n]
        elif len(f) < n:
            f += (n - len(f)) * '0'

    if integer_part == 0:
        m = -(decimal_bins.index(1) + 1)
        f = ''
        for i in range(decimal_bins.index(1) + 1, len(decimal_bins)):
            f += str(decimal_bins[i])
        if len(f) > n:
            f = f[:n]
        elif len(f) < n:
            f += (n - len(f)) * '0'

    return f, m
```

```python
def n_degree_taylor_derivative(formula, x, x_0, n, n_d):
    '''
    Given the analytic expression, x and x_0, calculate
    its n_d-th derivative based on n-th degree Taylor polynomial
    Note: Variable is defaulted to x

    Parameters:
    formula — analytic expression in sympy
    x — desired point
    x_0 — expansion point
    n — highest degree of polynomial
    n_d — degree of derivative

    Returns:
    n_d-th derivative based on n-th degree Taylor polynomial
    '''
    var_x = Symbol('x')
    acc = 0
    # Be careful, n+1 because it's up to n
    for i in range(n+1):
        acc += diff(formula, var_x, i).subs(var_x, x_0)/factorial(i)*(var_x-x_0)**i
    derivative = diff(acc, var_x, n_d)
    return float(derivative.subs(var_x, x))

x = Symbol('x')
n_degree_taylor_derivative(exp(x),3,0,2,1)
```

```python
from sympy import *
import numpy as np

def bin2dec(x):
    x = str(x)
    a = 0
    for i in range(2, len(x)):
        a += int(x[i]) * 2 ** (-i + 1)
    return a
```

Gap between FP: $\varepsilon \cdot 2^k$

```python
def n_degree_taylor(formula, x, x_0, n):
    '''
    Given the analytic expression, x and x_0, calculate
    its nth degree Taylor polynomial
    Note: Variable is defaulted to x
    '''
    var_x = Symbol('x')
    acc = 0
    # Be careful, n+1 because it's up to n
    for i in range(n+1):
        acc += diff(formula, var_x, i).subs(var_x, x_0)/factorial(i)*(x-x_0)**i
    return float(acc)
```

```python
# Rounding error decimal to binary FP
f, m = decimal_to_floating_point(3.34375, 10,100)
print(f,m)
(1+bin2dec('0.1010'))*2**m # 根据f 看清楚round up 还是round down
```

```
1010110000 1
```

```
3.25
```

```python
#f = lambda x : f0 + df0 * x + d2f0 * x**2/2
integration = lambda x: f0*x + df0 *x**2/2 + d2f0*x**3/6
I_approx = integration(h/2) - integration(0)
error = abs(I_approx-I)
```

积分代码

Question 1: Properties of bfloat16 Floating Point

The **bfloat16** floating-point system is often used in machine learning applications; to train machine learning algorithms, a large amount of calculation is necessary, but can be done without a high level of precision. This floating-point system uses only eight bits for the exponent and seven bits for the significand, which can make it significantly faster than double precision.

bfloat16 numbers are represented as

$$x = (-1)^s \times (1.b_1b_2b_3b_4b_5b_6b_7)_2 \times 2^m,$$

with $b_i \in \{0, 1\}$ and exponent $m \in [-126, 127]$.

What is the absolute value of the largest normalized number in this floating point system?

3.3895313892515355e+38 ✓ 100%

```python
k = 0
sum = a + 10 ** k
while sum != a:
    k -= 1
    sum = a + 10 ** k
k = k + 1
```

FP代码：a.k

```python
# Properties of bfloat16 Floating Point
n = 7
m = 127
x = "0." + 7 * "1"
(1+bin2dec(x))*2**m
```

标记得改

最大全为1
最小全为0

```python
def evaluate_fl_system(n, x, fl_lst):
    res_list=[]
    b = 2**(x-1)
    ma = 2**x-2
    l=1-b
    u=ma-b

    min_fp = 2**l
    max_fp = (2**(u+1))*(1-2**(-(n+1)))
    for i in fl_lst:
        temp = abs(i)
        if temp>max_fp:
            res_list.append("overflow")
        elif temp<min_fp:
            res_list.append("underflow")
        else:
            res_list.append("neither")
    return res_list
```

List 代码

Question 7: Floating point: exact representation

Consider a (binary) floating point system of the form $(-1)^s \times (1.b_1b_2b_3b_4)_2 \times 2^m$ where $s \in \{0, 1\}$ and $m \in \mathbb{Z} : m \in [-128, 127]$. What is the largest value, k, for which all of the integers in the range $[-k, k]$ are exactly representable in this floating point system?

32 ✓ 100%

```python
# exact representation
(1+bin2dec('0.1111'))*2**4
n = 4
2**(5+1) # 只有这行有用！ 看n
```

IEEE 双精度

exp = bin(a + 1023)[2:]

a 为往左移的小数点

Machine epsilon

bin2dec('0.0000001')

error bound

n	L.H.S. of (1)	is (1) satisfied?
1	0.5000	No
2	0.1667	No
3	0.0417	No
4	0.0083	No
5	0.0014	No
6	0.0002	No
7	0.00003	$\boxed{\text{YES}}$

FP:

$m \in [L, U]$

$p = n+1$

$2^{L} \sim 2^{U+1} \cdot (1-2^{-p})$

$\varepsilon = 2^{-n}$

IEEE 单:

$2^{-126} \sim 2^{128} \cdot (1-2^{-24})$ $\qquad 2^{-23} \cdot 2^{-126}$

$2^{-1022} \sim 2^{1024} (1-2^{-53})$ $\qquad 2^{-52} \times 2^{-1022}$

subnormal smallest.

$(0.00001) \times 2^{-4}$