In [ ]:

```python
# Package Imports
import numpy as np
import pandas as pd
```

Problem 1

Part a

In [ ]:

```python
def polynomial_evaluation(a,x):
    p = a[-1]*np.ones_like(x)
    for j in range(len(a)-2,-1,-1):
        p = a[j] + p*x

    return p

a = np.ones(51)
x = np.float64(1.00001)
result = polynomial_evaluation(a,x)
print(f"Result: {round(result,4)}")

true_value = (x**51 - 1)/(x-1)
print(f"Error: {true_value - result}")
```

```
Result: 51.0128
Error: -4.760636329592671e-12
```

Part b

b) i) $.1000111_{(2)}$

First, we use the shift property:

$$2^6 x = 1000 11.1000111$$
$$- \quad x = \qquad .1000111$$

$$(2^6 - 1)x = 100011_{(2)} = 35_{(10)}$$

$$\hookrightarrow 1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^5$$
$$1 + 2 + 32 = 35_{(10)}$$

$$x = \frac{35_{(10)}}{(64-1)_{10}} = \frac{35}{63} = \boxed{\frac{5}{9}}$$

ii) $\frac{5}{9} = .\overline{5}_{(10)}$

$$.5 \times 16 = 8.0 \Rightarrow 8$$
$$.5 \times 16 = 8.0 \Rightarrow 8$$
$$\vdots$$

so we have $\boxed{.\overline{8}_{(16)}}$

Problem 2

2. 20.1 to double precision FP

Integer Part:

$20 = 16 + 4 = 2^4 + 2^2 = 10100_{(2)}$

Fraction Part:

$.1 \times 2 = .2 + 0$
$.2 \times 2 = .4 + 0$
$.4 \times 2 = .8 + 0$
$.8 \times 2 = .6 + 1$
$.6 \times 2 = .2 + 1$
$.2 \times 2 = .4 + 0$
$.4 \times 2 = .8 + 0$
$\vdots$

Repeats

Hence, we have $10100.0\overline{0011}$

Left - Justify

$1.0100\,0\overline{0011} \times 2^4$

- sign $+ \rightarrow s = 0$
- Exponent:

$4 = 1027 - 1023 = (1024 + 2 + 1) - 1023$

$e \rightarrow 2^{10} + 2^1 + 2^0 = 100,0000,0011_{(2)}$

$$10100.0\overline{0011}_{(2)} =$$

$$1.[\underset{\uparrow \atop s}{0}\underbrace{100,0000,0011}_{e},\underbrace{0100,0001,1001,1001,1001,1001,1001,1001,1001,1001,1001,1001,1001}_{mantissa}],1001$$

we round up the last digit by 1 since the first truncated bit is 1 and all known bits to the right of that are not 0

$$\boxed{\approx 1.[\underset{\uparrow \atop s}{0}\underbrace{100,0000,0011}_{e},\underbrace{0100,0001,1001,1001,1001,1001,1001,1001,1001,1001,1001,1001,1010}_{mantissa}]}$$

Problem 3

Part a

3, a) By Taylor Expansion:

$$\sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + O(x^9)$$

we plug this into our expression:

$$x - \sin x \approx x - \left(x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}\right)$$

$$= \frac{x^3}{3!} - \frac{x^5}{5!} - \frac{x^7}{7!}$$

we will also use nested multiplication:

$$x - \sin x \approx x^3\left(\frac{1}{6} + x^2\left(-\frac{1}{120} + \frac{1}{5040}x^2\right)\right)$$

Part b

In [ ]:

```python
def loss_of_significance(x):
    return x**3/6 - x**5/120 + x**7/5040

vals = np.linspace(1,33,17)
x = np.float64(2.**-vals)

test = x - np.sin(x)
result = loss_of_significance(x)

# Use Horner's algorithm
a = np.array([0,0,0,1/6,0,-1/120,1/5040])
result_nested_eval = polynomial_evaluation(a,x)

pd.DataFrame({'x': [f'2^-{int(val)}' for val in vals],
              'Test': test, 'Result': result,
              'Nested Result': result_nested_eval})
```

Out[ ]:

	x	Test	Result	Nested Result
0	2^-1	2.057446e-02	2.057447e-02	2.057602e-02
1	2^-3	3.252666e-04	3.252666e-04	3.252673e-04
2	2^-5	5.086015e-06	5.086015e-06	5.086015e-06
3	2^-7	7.947262e-08	7.947262e-08	7.947262e-08
4	2^-9	1.241763e-09	1.241763e-09	1.241763e-09
5	2^-11	1.940255e-11	1.940255e-11	1.940255e-11
6	2^-13	3.031649e-13	3.031649e-13	3.031649e-13
7	2^-15	4.736950e-15	4.736952e-15	4.736952e-15
8	2^-17	7.401459e-17	7.401487e-17	7.401487e-17
9	2^-19	1.156412e-18	1.156482e-18	1.156482e-18
10	2^-21	1.805239e-20	1.807004e-20	1.807004e-20
11	2^-23	2.779327e-22	2.823443e-22	2.823443e-22
12	2^-25	3.308722e-24	4.411630e-24	4.411630e-24
13	2^-27	0.000000e+00	6.893172e-26	6.893172e-26
14	2^-29	0.000000e+00	1.077058e-27	1.077058e-27
15	2^-31	0.000000e+00	1.682903e-29	1.682903e-29
16	2^-33	0.000000e+00	2.629536e-31	2.629536e-31

Part c

c) I believe that this method does not involve severe loss of significance b/c we can avoid subtracting numbers that are very similar like was the case for $y = x - \sin x$.

For example $y(0) = 0 - 0$ so $y(x)$ for a small value of x will result in a loss of significance, but not for our modified version

Problem 4

4. Let $F = g + s \Rightarrow F'' = g'' + s''$

$$\overset{D}{\int_a^b (F''(x))^2\,dx} = \int_a^b (g''(x) + s''(x))^2\,dx$$

$$= \overset{A}{\int_a^b (g''(x))^2\,dx} + \overset{B}{\int_a^b (s''(x))^2\,dx} + 2\overset{C}{\int_a^b s''(x)\,g''(x)\,dx}$$

since $A \geq 0$, we must only show that $C \geq 0$

to Prove $B \leq D$

$$C = \int_a^b s''(x)\,g''(x)\,dx = u \cdot v \Big|_a^b - \int_a^b v\,du$$

Let $u = s''(x)$ $\quad du = s'''(x)\,dx$

$v = g'(x)$ $\quad dv = g''(x)\,dx$

$$= s''(x) \cdot g'(x) \Big|_a^b - \int_a^b g'(x)\,s'''(x)\,dx$$

$s''' \equiv$ constant since s is cubic

$$= \left[s''(t_n)\,g'(t_n) - s''(t_0)\,g'(t_0)\right] - s''' \int_{t_0}^{t_n} g'(x)\,dx$$

By FTOC

$g(b) - g(a)$

$$= \overset{a}{\left[s''(t_n)\,g'(t_n) - s''(t_0)\,g'(t_0)\right]} - \overset{b}{s'''\left[g(t_n) - g(t_0)\right]}$$

(a)

Let $S(x) = \begin{cases} S_0(x) & x \in [t_0, t_1] \\ \vdots & \\ S_{n-1}(x) & x \in [t_{n-1}, t_n] \end{cases}$

Thus $S''(t_n) = \underbrace{S_0''(t_n)}_{t_n \notin [t_0, t_1] \to 0} + \underbrace{\dots}_{t_n \notin [t_1, t_{n-1}] \to 0} + \underbrace{S_{n-1}''(t_n)}_{t_n \in [t_{n-1}, t_n]}$

$= S_{n-1}''(t_n) = 0$ by natural spline condition

Similarly, $S''(t_0) = S_0''(t_0) = 0$

Hence, (a) $= 0$

(b)

$g(t_n) = f(t_n) - S(t_n) = 0$ by 1st property of spline

Similarly, $g(t_0) = f(t_0) - S(t_0) = 0$

Hence, (b) $= 0$

Finally, (c) $= 0$

Thus (D) $=$ (A) $+$ (B) where (A) ≥ 0

$\Rightarrow$ (B) $\leq$ (D)

$$\Rightarrow \int_a^b (S''(x))^2 \, dx \leq \int_a^b (f''(x))^2 \, dx \quad \square$$

Problem 5

5. a) $d_0 + x(d_1 + x(d_2 + \dots + x d_n)\dots)$

We can see n additions and n multiplications where n is the polynomial order

b) Machine Epsilon is the distance between 1 and the largest floating point number smaller than 1. This represents the smallest distinguishable size between numbers on the unit (1) scale.

c) False

d) $\epsilon_{mach} = 2^{-52}$

This is determined by $1 - .999\dots_{(10)}$

$$= 1 - (.111\dots10)_{(2)} = 2^{-52}$$

since there are 52 bits for the mantissa

e) The smallest number that can be represented by IEEE double precision is 2^{-1074}

this comes from $\underset{s}{0}\,\underbrace{000,0000,0000}_{e},\underbrace{0000,0000,\dots,0001}_{\text{mantissa}}$

which is $(-1)^0 \cdot (1.0\dots01) \cdot 2^{-1022}$

$$= 2^{-52} \cdot 2^{-1022} = 2^{-1074}$$

f) True

3) As was the case in Problem 3:

$$y(x) = x - \sin(x)$$

$$@\ x \approx 0$$

$$y(2^{-33}) \approx 2.629 \times 10^{-31}$$

but when evaluated, the two numbers are so close that significance/accuracy are lost and the computer returns 0.

In this situation we can avoid this error by expanding sin x in Taylor series to avoid subtracting very similar numbers (see question 3), but that is not always possible