	Marwadi University **Faculty of Engineering & Technology** **Department of Information and Communication Technology**	
Subject: Programming With Python (01CT1309)	**Aim:** Analysis of Discrete-Time Signals Using Z-Transform	
Experiment No: 17	**Date:**	**Enrollment No:92510133028**

Aim: Analysis of Discrete-Time Signals Using Z-Transform

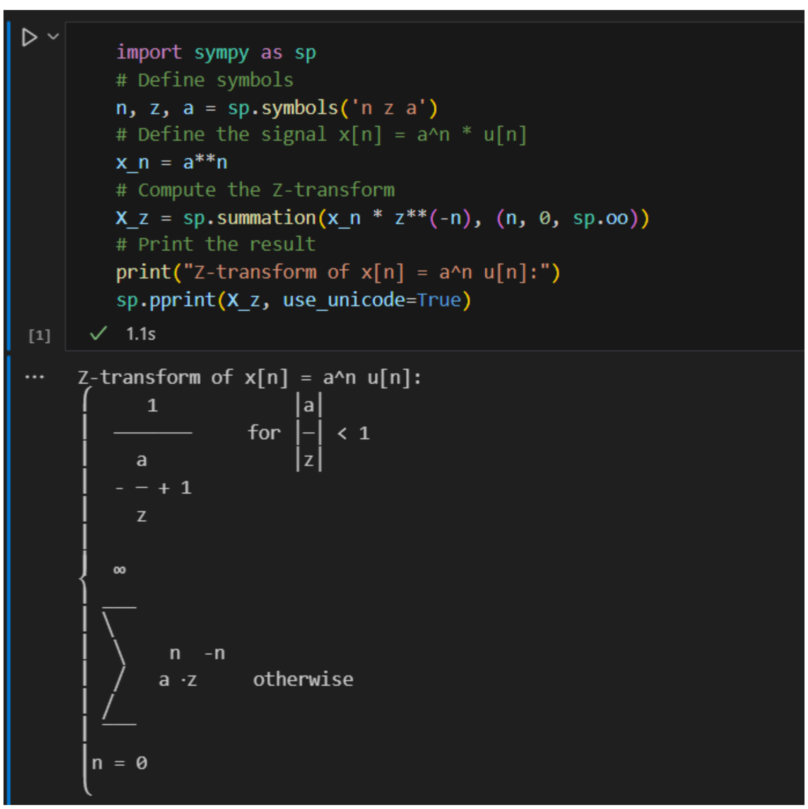
IDE:

Install Library

pip install sympy

Example 1:

```
import sympy as sp
# Define symbols
n, z, a = sp.symbols('n z a')
# Define the signal x[n] = a^n * u[n]
x_n = a**n
# Compute the Z-transform
X_z = sp.summation(x_n * z**(-n), (n, 0, sp.oo))
# Print the result
print("Z-transform of x[n] = a^n u[n]:")
sp.pprint(X_z, use_unicode=True)
```

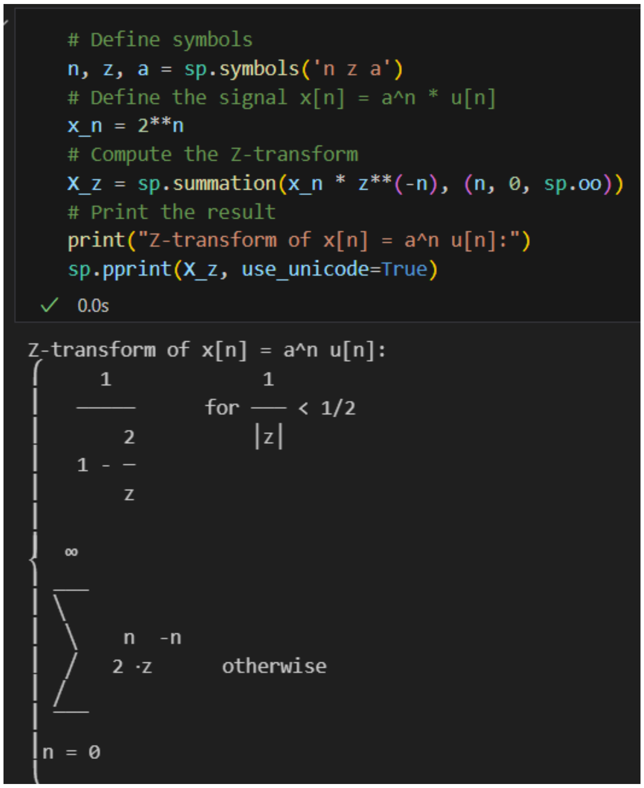


```
import sympy as sp
# Define symbols
n, z, a = sp.symbols('n z a')
# Define the signal x[n] = a^n * u[n]
x_n = a**n
# Compute the Z-transform
X_z = sp.summation(x_n * z**(-n), (n, 0, sp.oo))
# Print the result
print("Z-transform of x[n] = a^n u[n]:")
sp.pprint(X_z, use_unicode=True)
[1] ✓ 1.1s

Z-transform of x[n] = a^n u[n]:
⎧    1             │a│
⎪  ──────     for  │─│ < 1
⎪    a             │z│
⎪  - ─ + 1
⎪    z
⎪
⎨  ∞
⎪ ___
⎪ ╲
⎪  ╲    n  -n
⎪  ╱   a ⋅z     otherwise
⎪ ╱
⎪ ‾‾‾
⎩n = 0
```

Example 2:

```
# Define symbols
n, z, a = sp.symbols('n z a')
# Define the signal x[n] = a^n * u[n]
x_n = 2**n
# Compute the Z-transform
X_z = sp.summation(x_n * z**(-n), (n, 0, sp.oo))
# Print the result
print("Z-transform of x[n] = a^n u[n]:")
sp.pprint(X_z, use_unicode=True)
```



```
# Define symbols
n, z, a = sp.symbols('n z a')
# Define the signal x[n] = a^n * u[n]
x_n = 2**n
# Compute the Z-transform
X_z = sp.summation(x_n * z**(-n), (n, 0, sp.oo))
# Print the result
print("Z-transform of x[n] = a^n u[n]:")
sp.pprint(X_z, use_unicode=True)
✓ 0.0s

Z-transform of x[n] = a^n u[n]:
⎧    1              1
⎪  ─────      for ─── < 1/2
⎪      2          │z│
⎪  1 - ─
⎪      z
⎪
⎨ ∞
⎪ ___
⎪ ╲
⎪  ╲    n  -n
⎪  ╱   2 ⋅z       otherwise
⎪ ╱
⎪ ‾‾‾
⎪n = 0
⎩
```

Example 3:

import sympy as sp

Define symbols

n, z = sp.symbols('n z')

Define the unit step signal u[n]

u_n = 1

Compute the Z-transform

U_z = sp.summation(u_n * z**(-n), (n, 0, sp.oo))

Print the result

print("Z-transform of the unit step signal u[n]:")

sp.pprint(U_z, use_unicode=True)

```
import sympy as sp
# Define symbols
n, z = sp.symbols('n z')
# Define the unit step signal u[n]
u_n = 1
# Compute the Z-transform
U_z = sp.summation(u_n * z**(-n), (n, 0, sp.oo))
# Print the result
print("Z-transform of the unit step signal u[n]:")
sp.pprint(U_z, use_unicode=True)
```

✓ 0.0s

```
Z-transform of the unit step signal u[n]:
⎧   1           1
⎪ ─────    for ─── < 1
⎪     1        │z│
⎪ 1 - ─
⎪     z
⎪
⎨  ∞
⎪ ___
⎪ ╲
⎪  ╲    -n
⎪  ╱   z      otherwise
⎪ ╱
⎪ ‾‾‾
⎪n = 0
⎩
```

Example 4:

import sympy as sp

Define symbols

n, z, alpha = sp.symbols('n z alpha')

Define the signal x[n] = exp(alpha * n) * u[n]

x_n = sp.exp(alpha * n)

Compute the Z-transform

X_z = sp.summation(x_n * z**(-n), (n, 0, sp.oo))

Print the result

print("Z-transform of x[n] = exp(alpha * n) u[n]:")

sp.pprint(X_z, use_unicode=True)

```
import sympy as sp
# Define symbols
n, z, alpha = sp.symbols('n z alpha')
# Define the signal x[n] = exp(alpha * n) * u[n]
x_n = sp.exp(alpha * n)
# Compute the Z-transform
X_z = sp.summation(x_n * z**(-n), (n, 0, sp.oo))
# Print the result
print("Z-transform of x[n] = exp(alpha * n) u[n]:")
sp.pprint(X_z, use_unicode=True)
```

✓ 0.0s

```
Z-transform of x[n] = exp(alpha * n) u[n]:
  ∞
 ___
 ╲
  ╲    -n  α⋅n
  ╱   z  ⋅ℯ
 ╱
 ‾‾‾
n = 0
```

Example 5:

import sympy as sp

Define symbols

n, z = sp.symbols('n z')

Define the finite sequence x[n] = {1, 2, 3}

x_n = [1, 2, 3]

Compute the Z-transform manually

X_z = sum(x_n[i] * z**(-i) for i in range(len(x_n)))

Print the result

print("Z-transform of the finite sequence {1, 2, 3}:")

sp.pprint(X_z, use_unicode=True)

```
import sympy as sp
# Define symbols
n, z = sp.symbols('n z')
# Define the finite sequence x[n] = {1, 2, 3}
x_n = [1, 2, 3]
# Compute the Z-transform manually
X_z = sum(x_n[i] * z**(-i) for i in range(len(x_n)))
# Print the result
print("Z-transform of the finite sequence {1, 2, 3}:")
sp.pprint(X_z, use_unicode=True)
```

✓ 0.0s

```
Z-transform of the finite sequence {1, 2, 3}:
    2   3
1 + ─ + ──
    z    2
        z
```

Example 6

import sympy as sp

Define symbols

n, z, omega = sp.symbols('n z omega')

Define the sinusoidal sequence x[n] = sin(omega * n) * u[n]

x_n = sp.sin(omega * n)

Compute the Z-transform

X_z = sp.summation(x_n * z**(-n), (n, 0, sp.oo))

# Print the result
print("Z-transform of x[n] = sin(omega * n) u[n]:")
sp.pprint(X_z, use_unicode=True)

```python
import sympy as sp
# Define symbols
n, z, omega = sp.symbols('n z omega')
# Define the sinusoidal sequence x[n] = sin(omega * n) * u[n]
x_n = sp.sin(omega * n)
# Compute the Z-transform
X_z = sp.summation(x_n * z**(-n), (n, 0, sp.oo))
# Print the result
print("Z-transform of x[n] = sin(omega * n) u[n]:")
sp.pprint(X_z, use_unicode=True)
```

✓ 0.0s

```
Z-transform of x[n] = sin(omega * n) u[n]:
  ∞
 ___
 \
  \     -n
  /    z  ⋅sin(n⋅ω)
 /__
n = 0
```

Marwadi University (Marwadi Chandarana Group), NAAC A+	Marwadi University Faculty of Engineering & Technology Department of Information and Communication Technology	
Subject: Programming With Python (01CT1309)	**Aim:** Analysis of Discrete-Time Signals Using Z-Transform	
Experiment No: 17	**Date:**	**Enrollment No:92510133028**

Post Lab Exercise:

- Using Python, compute the Z-transform of the sequence $x[n] = 3^n u[n]$.

```python
# ⍰Using Python, compute the Z-transform of the sequence . x[n] = 3^n * u[n]
import sympy as sp
# Define symbols
n, z = sp.symbols('n z')
# Define the signal x[n] = 3^n * u[n]
x_n = 3**n
# Compute the Z-transform
X_z = sp.summation(x_n * z**(-n), (n,
    0, sp.oo))
# Print the result
print("Z-transform of x[n] = 3^n u[n]:")
sp.pprint(X_z, use_unicode=True)
```

✓ 0.0s

```
Z-transform of x[n] = 3^n u[n]:
⎧    1             1
⎪  ─────      for ─── < 1/3
⎪      3          │z│
⎪  1 - ─
⎪      z
⎪
⎨  ∞
⎪ ___
⎪ ╲
⎪  ╲    n  -n
⎪  ╱   3 ·z      otherwise
⎪ ╱
⎪ ‾‾‾
⎩n = 0
```

- Using Python, compute the Z-transform of the sequence $x[n] = \cos(wn)u[n]$.

```python
# Using Python, compute the Z-transform of the sequence x[n] = cos(wn) * u[n]
import sympy as sp
# Define symbols
n, z, omega = sp.symbols('n z omega')
# Define the sinusoidal sequence x[n] = cos(omega * n) * u[n]
x_n = sp.cos(omega * n)
# Compute the Z-transform
X_z = sp.summation(x_n * z**(-n), (n, 0, sp.oo))
# Print the result
print("Z-transform of x[n] = cos(omega * n) u[n]:")
sp.pprint(X_z, use_unicode=True)
```

✓ 0.0s

```
Z-transform of x[n] = cos(omega * n) u[n]:
  ∞
 ____
 ╲
  ╲
   ╲    -n
   ╱   z  ·cos(n⋅ω)
  ╱
 ╱
 ‾‾‾‾
n = 0
```

Github: https://github.com/keshvi1234/PWP_experiment