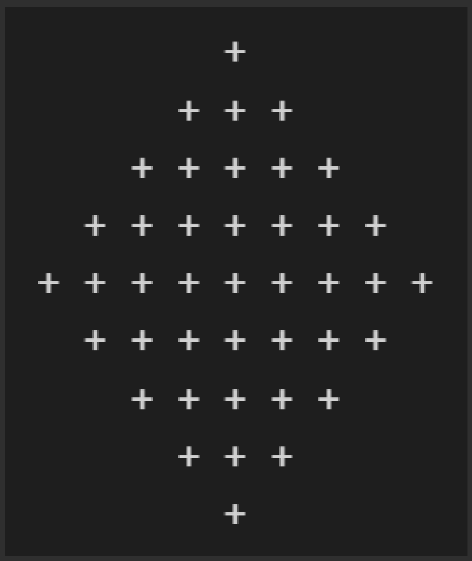
# Mastering f-Strings in Python

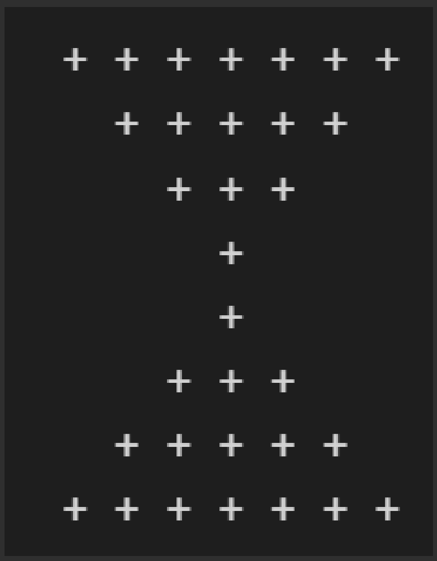
Cahya Alkahfi

sainsdata.id

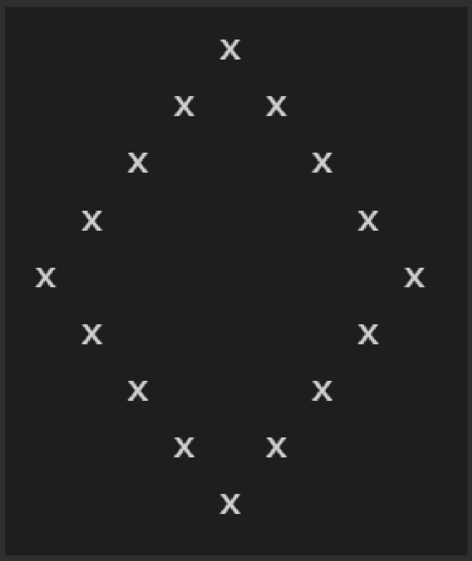
## Table of contents



```
        +
      + + +
    + + + + +
  + + + + + + +
+ + + + + + + + +
  + + + + + + +
    + + + + +
      + + +
        +
```



```
+ + + + + + +
  + + + + +
    + + +
      +
      +
    + + +
  + + + + +
+ + + + + + +
```



```
        x
      x   x
    x       x
  x           x
x               x
  x           x
    x       x
      x   x
        x
```

## What are f-Strings?

f-Strings, short for formatted string literals, were introduced in Python 3.6. They provide a concise and readable way to embed expressions inside string literals using curly braces {}.

f-Strings are designed to be more readable and less error-prone compared to previous string formatting methods. They allow embedding of Python expressions directly within strings, making the code cleaner and easier to understand.

### Benefits of Using f-Strings

**Readability:** improve code readability by reducing the verbosity associated with other formatting methods. Embedding expressions directly within strings makes the intent of the code clearer.

**Efficiency:** more efficient than both the % operator and str.format() method. They are faster because they are evaluated at runtime, reducing the overhead of additional function calls.

**Flexibility:** provide great flexibility in embedding various types of expressions, including variables, arithmetic operations, and even function calls. They also support advanced formatting options like alignment, padding, and precision control.

**Ease of Use:** simplify string formatting tasks and reduce the likelihood of errors. They eliminate the need for placeholders and explicit argument indexing, making the code more intuitive and less prone to mistakes.

## Creating an f-String

f-Strings provide a simple and intuitive way to embed expressions inside string literals. To create an f-String, prefix a string with the letter f or F, followed by the string in either single, double, or triple quotes, with every expression or operation enclosed in curly braces {}.

```
name = "John Doe"
greeting = f"Hello, I am {name}"
print(greeting)
```

```
Hello, I am John Doe
```

```
first_name = "John"
last_name = "Doe"
age = 50
full_name = f"{first_name} {last_name}"
greeting = f"Hello, my name is {full_name} and I am {age} years old"
print(greeting)
```

```
Hello, my name is John Doe and I am 50 years old
```

```
radius = 5
area = f"The area of the circle is {3.14 * radius ** 2:.2f} square units."
print(area)
```

```
The area of the circle is 78.50 square units.
```

In Python 3.8 and later, we can use "self-documenting expressions" to display expressions and their results directly by using the equals sign (=) within the f-String expression.

```
a = 10
b = 0.125

print(f"{a + b = :.2f}, {a - b = :.2f}, {a * b = }")
```

```
 a + b = 10.12, a - b = 9.88, a * b = 1.25
```

## Formatting Numbers

```python
rate = 0.84579

# 4 decimal places
fmt_rate_1 = f"Rate to 4 decimal places is {rate:.4f}"
print(fmt_rate_1)

# 2 decimal places percentage
fmt_rate_2 = f"Rate as a percentage to 2 decimal places is {rate:.2%}"
print(fmt_rate_2)
```

```
Rate to 4 decimal places is 0.8458
Rate as a percentage to 2 decimal places is 84.58%
```

```python
distance = 1384400000
fmt_distance_1 = f"Distance: {distance:,.0f}"  # comma separator
print(fmt_distance_1)

fmt_distance_2 = f"Distance: {distance:_.0f}"  # underscore separator
print(fmt_distance_2)
```

```
Distance: 1,384,400,000
Distance: 1_384_400_000
```

```python
speed = 299_792_458

# Format with standard scientific notation (lowercase 'e')
fmt_speed_1 = f"Speed of light is {speed:.2e} m/s"
print(fmt_speed_1)

# Format with uppercase scientific notation (uppercase 'E')
fmt_speed_2 = f"Speed of light is {speed:.4E} m/s"
print(fmt_speed_2)
```

```
Speed of light is 3.00e+08 m/s
Speed of light is 2.9979E+08 m/s
```

## Aligning Strings

```python
data = [225000, 5000, 15000000]

print("Left-aligned:")
for dt in data:
    print(f"{dt:<8}")    # Left-align 8 characters (<8)

print("\nRight-aligned (by 10 chars):")
for dt in data:
    print(f"{dt:>10}")   # Right-align 10 characters (>10)
print("----------")

print("\nCenter-aligned (by 13 chars):")
for dt in data:
    print(f"{dt:^13}")   # Center-align 13 characters (^13)
print("-------------")
```

```
Left-aligned:
225000
5000
15000000

Right-aligned (by 10 chars):
    225000
      5000
  15000000
----------

Center-aligned (by 13 chars):
   225000
    5000
  15000000
-------------
```

## Padding Strings

As before, but we can fill the empty space with any character.

```python
data = [225000, 5000, 15000000]

print("Right padding with '*' (8 chars):\n")
for dt in data:
    print(f"{dt:*<8}")

print("\nLeft padding with 'X' (10 chars):\n")
for dt in data:
    print(f"{dt:X>10}")

print("\nLeft and right padding with '_' (12 chars):\n")
for dt in data:
    print(f"{dt:_^12}")
```

```
Right padding with '*' (8 chars):

225000**
5000****
15000000

Left padding with 'X' (10 chars):

XXXX225000
XXXXXX5000
XX15000000

Left and right padding with '_' (12 chars):

___225000___
____5000____
__15000000__
```

## Date and Time Formatting

Using f-Strings along with the datetime module allows us to format dates and times flexibly and conveniently.

```python
from datetime import datetime

# Get the current date and time
now = datetime.now()

# Example 1: Full date and time
formatted_now_1 = f"{now:%Y-%m-%d %H:%M:%S}"
print(f"Full date and time: {formatted_now_1}")

# Example 2: Date only
formatted_now_2 = f"{now:%Y-%m-%d}"
print(f"Date only: {formatted_now_2}")

# Example 3: Time only
formatted_now_3 = f"{now:%H:%M:%S}"
print(f"Time only: {formatted_now_3}")

# Example 4: Custom format
formatted_now_4 = f"{now:%A, %B %d, %Y at %I:%M %p}"
print(f"Cust. format: {formatted_now_4}")
```

```
Full date and time: 2024-08-04 19:31:45
Date only: 2024-08-04
Time only: 19:31:45
Cust. format: Sunday, August 04, 2024 at 07:31 PM
```

## Special Characters & Escape Sequences

Although these are not exclusive features of f-Strings, f-Strings make it easy to include special characters like quotes, backslashes, and newline characters by using escape sequences or raw string literals.

```python
item = "Laptop"
specs = ["15\" screen", "16GB RAM", "512GB SSD"]

output = f"Product: {item}\nSpecs:\n\t{specs[0]}\n\t{specs[1]}\n\t{specs[2]}"
print(output)
```

```
Product: Laptop
Specs:
        15" screen
        16GB RAM
        512GB SSD
```

```python
# using raw string
path1 = r'"C:\Users\SainsData\Documents\Report1.txt"'

# using escape character (\" and \\)
path2 = "\"C:\\Users\\SainsData\\Documents\\Report2.txt\""

output = f"The file path is:\n > {path1}\n > {path2}"
print(output)
```

```
The file path is:
 > "C:\Users\SainsData\Documents\Report1.txt"
 > "C:\Users\SainsData\Documents\Report2.txt"
```

## Multiline f-Strings (1/3)

Multiline f-Strings use triple quotes (''' or """) to enclose the string, and we can include expressions inside curly braces {} just like with single-line f-Strings. We can also use join Method to create a list of items separated by newline characters.

```
width = 10
height = 5

area = f"""
The dimensions of the rectangle are:
> Width: {width} units
> Height: {height} units
> Area: {width * height} square units
"""

print(area)
```

```
The dimensions of the rectangle are:
> Width: 10 units
> Height: 5 units
> Area: 50 square units
```

```
items = ["apple", "banana", "cherry"]

shopping_list = f"""
Shopping List:
{'- ' + '\n- '.join(items)}
"""

print(shopping_list)
```

```
Shopping List:
- apple
- banana
- cherry
```

## Multiline f-Strings (2/3)

```python
items = {
    "Laptop": 999.99,
    "Smartphone": 499.99,
    "Tablet": 299.99,
    "Headphones": 199.99,
    "Charger": 29.99
}

output = f"""
{"-"*24}
{"Product":<12} | {"Price":>7}
{"-"*24}
"""

for item, price in items.items():
    output += f"{item:<12} | ${price:>7.2f}\n"

output+=f"{'-'*24}"

print(output)
```

```
------------------------
Product      |   Price
------------------------
Laptop       | $ 999.99
Smartphone   | $ 499.99
Tablet       | $ 299.99
Headphones   | $ 199.99
Charger      | $  29.99
------------------------
```

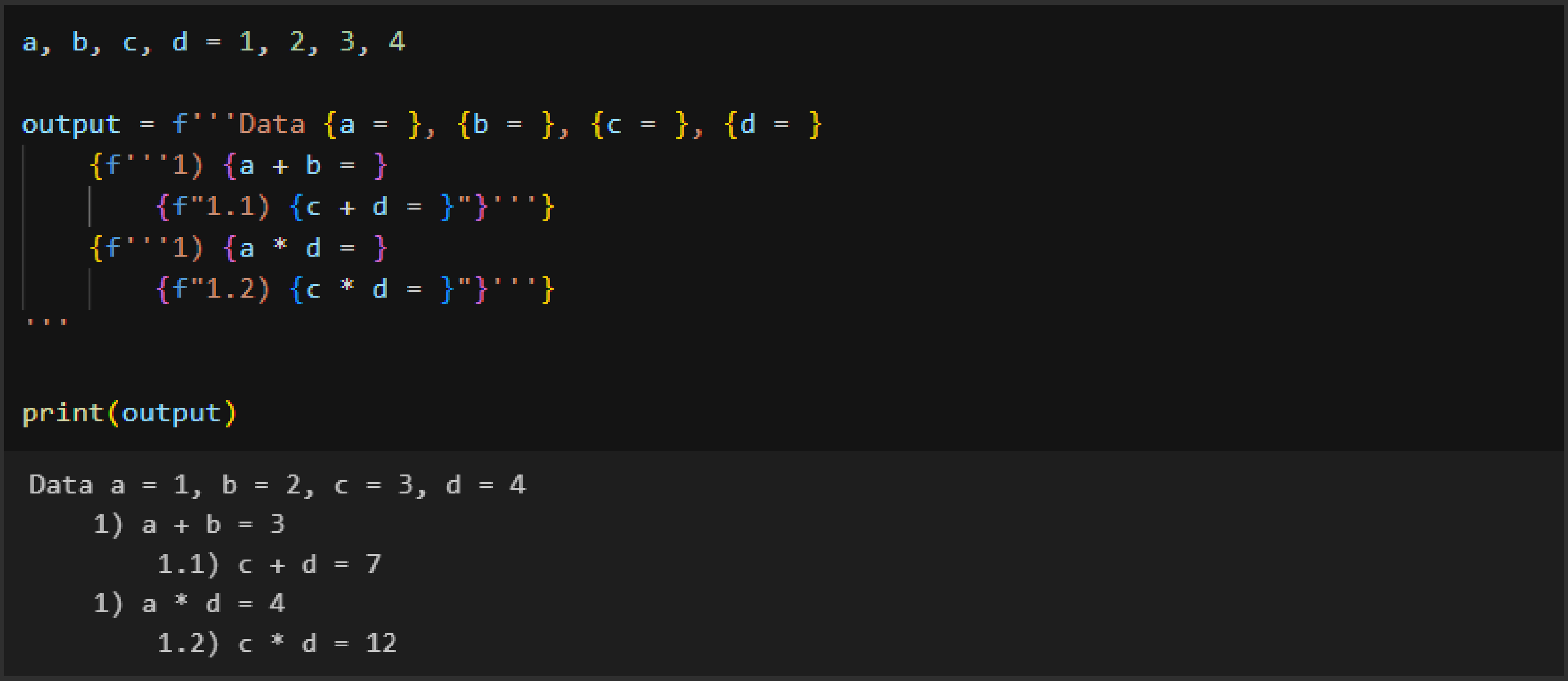
## Multiline f-Strings (3/3)

```python
send_email = lambda info: f"""
To: {info['recipient']}
Subject: {info['subject']}

Dear {info['username']},

Welcome to our service! We are thrilled to have you on board.

Best regards,
The Team
"""

my_info = {
    'recipient': "johndoe@example.com",
    'subject': "Welcome to Our Service",
    'username': "John Doe"}

print(send_email(my_info))
```

```
To: johndoe@example.com
Subject: Welcome to Our Service

Dear John Doe,

Welcome to our service! We are thrilled to have you on board.

Best regards,
The Team
```

## Nested f-Strings

If necessary, we can also use nested f-Strings, where an f-String contains another f-String within its expression.



```python
a, b, c, d = 1, 2, 3, 4

output = f'''Data {a = }, {b = }, {c = }, {d = }
    {f'''1) {a + b = }
        {f"1.1) {c + d = }"}'''}
    {f'''1) {a * d = }
        {f"1.2) {c * d = }"}'''}
'''

print(output)
```

```
Data a = 1, b = 2, c = 3, d = 4
    1) a + b = 3
        1.1) c + d = 7
    1) a * d = 4
        1.2) c * d = 12
```

## Bonus (Star Patterns using f-Strings)

```python
start = 1; stop = 14; step = 2

for i in range(start, stop, step):
    print(f"{'* ' * i:^{2*stop}}")
```

```
             *
           * * *
         * * * * *
       * * * * * * *
     * * * * * * * * *
   * * * * * * * * * * *
 * * * * * * * * * * * * *
```

```python
start = 1; stop = 14; step = 2

for i in range(start, stop, step):
    print(f"{'* ' * (stop - i):^{2*stop}}")
```

```
 * * * * * * * * * * * * *
   * * * * * * * * * * *
     * * * * * * * * *
       * * * * * * *
         * * * * *
           * * *
             *
```

## Bonus (Star Patterns using f-Strings)

```python
start = 1; stop = 8; step = 1

for i in range(start, stop, step):
    print(f"{'x  ' * i:<{2*stop}}")
```

```
x
x  x
x  x  x
x  x  x  x
x  x  x  x  x
x  x  x  x  x  x
x  x  x  x  x  x  x
```

```python
start = 1; stop = 8; step = 1

for i in range(start, stop, step):
    print(f"{'x  ' * (stop - i):<{2*stop}}")
```

```
x  x  x  x  x  x  x
x  x  x  x  x  x
x  x  x  x  x
x  x  x  x
x  x  x
x  x
x
```

## Bonus (Star Patterns using f-Strings)

```python
start = 1; stop = 18; step = 2

for i in range(start, stop, step):
    print(f"{'+ ' * (stop//2 - abs(i - stop//2)):^{2*stop}}")
```

```
                 + 
               + + + 
             + + + + + 
           + + + + + + + 
         + + + + + + + + + 
           + + + + + + + 
             + + + + + 
               + + + 
                 + 
```

```python
start = 1; stop = 17; step = 2

for i in range(start, stop, step):
    print(f"{'+ ' * abs(i - stop//2):^{2*stop}}")
```

```
          + + + + + + + 
            + + + + + 
              + + + 
                + 
                + 
              + + + 
            + + + + + 
          + + + + + + + 
```

## Bonus (Star Patterns using f-Strings)

```python
start=1; stop=9; step=2

for i in range(start, 2*stop, step):
    spaces = abs(stop - i)
    x = stop - spaces
    print(f"{' ' * spaces}x{' '*(2*x - 3)}{'x' if x > 1 else ''}")
```

```
    x
   x x
  x   x
 x     x
x       x
 x     x
  x   x
   x x
    x
```

THANK YOU

sainsdata.id