COSC 1306 - Prog for Non-Majors

I/O Processing-2

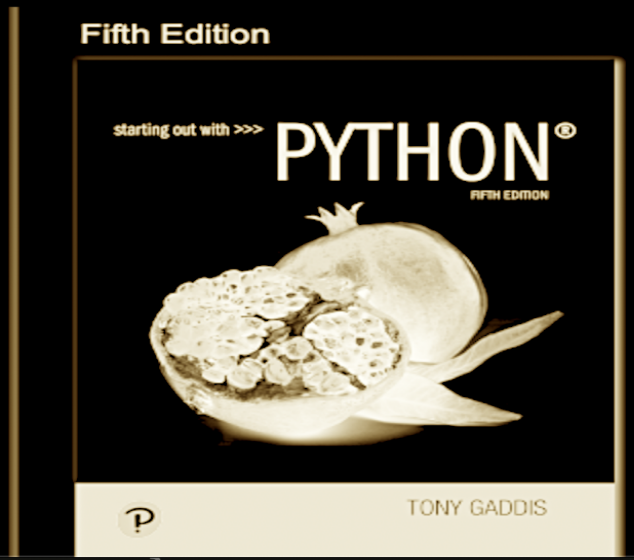
Dr. Mohan

McMurry University

September 11, 2023

Lesson Topics Overview

- String Concatenation.
- More About The print Function.
- Displaying Formatted Output.
- Named Constants.



Chapter 2

Input, Processing, and Output

String Concatenation

- Concatenation: To append one string to the end of another string.
- Use the + operator to concatenate strings.

```
message = 'Hello ' + 'World'
print(message)
```

The output is: Hello World

String Concatenation (Cont'd)

- You can use string concatenation to break up a long string literal.

```
print('Enter the amount of ' +
      'sales for each day and ' +
      'press Enter.')
```

This statement will display the following:

Enter the amount of sales for each day and press Enter.

Implicit Concatenation

- Two or more string literals written adjacent to each other are implicitly concatenated into a single string.

```
my_str = 'one' 'two' 'three'
print(my_str)
```

The output is: onetwothree

Implicit Concatenation (Cont'd)

```
print('Enter the amount of '
      'sales for each day and '
      'press Enter.')
```

This statement will display the following:

Enter the amount of sales for each day and press Enter.

More About The print Function

- **print function** displays the line of output with newline character at end of printed data and space as the item separator.
- Special argument end='delimiter' causes print to place delimiter at end of data instead of newline character.
- Special argument sep='delimiter' causes print to use delimiter as item separator.

More About The print Function (Cont'd)

- Special characters appearing in string literal.
- Preceded by backslash (\).
- Examples: newline (\n), horizontal tab (\t).
- Treated as commands embedded in string.

More About The print Function (Cont'd)

- Special characters appearing in string literal.
- Preceded by backslash (\).
- Examples: newline (\n), horizontal tab (\t).
- Treated as commands embedded in string.

Formatted Output & F-strings

- An f-string is a special type of string literal that is prefixed with the letter f.

  print(f'Hello World')

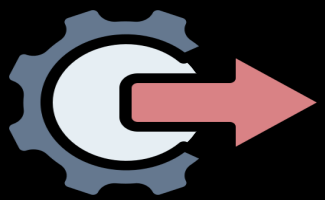
  $\rightarrow$ **Hello world**

- F-strings support placeholders for variables:

  name = 'Johnny'

  print(f'Hello name.')

  $\rightarrow$ **Hello Johnny.**



Formatted Output & F-strings (Cont'd)

- Placeholders can also be expressions that are evaluated.

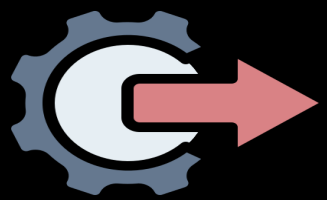
  print(f'The value is 10 + 2.')

  $\rightarrow$ The value is 12.

  val = 10

  print(f'The value is val + 2.')

  $\rightarrow$ The value is 12.



Formatted Output & F-strings (Cont'd)

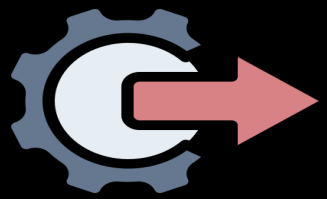
- Format specifiers can be used with placeholders.

  num = 123.456789

  print(f'num:.2f')

  $\rightarrow$ **123.46**

- **.2f means:** round the value to 2 decimal places and display the value as a floating-point number.



Formatted Output & F-strings (Cont'd)

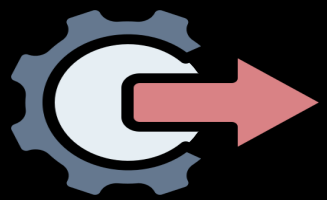
- Format specifiers can be used with placeholders.

```
num = 1000000.00
print(f'num:,.2f')
```

→ 1,000,000.00

```
discount = 0.5
print(f'discount:.0%')
```

→ 50%



Formatted Output & F-strings (Cont'd)

- Format specifiers can be used with placeholders.

num = 123456789

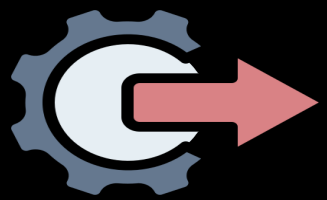
print(f'num:,d')

$\rightarrow$ **123,456,789**

num = 12345.6789

print(f'num:.2e')

$\rightarrow$ **1.23e+04**



Formatted Output & F-strings (Cont'd)

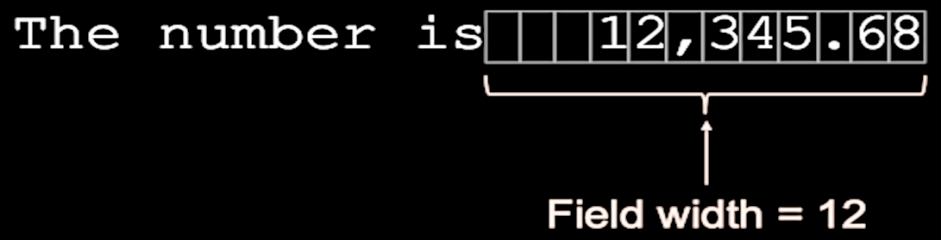
- Specifying a minimum field width:

  num = 12345.6789

  print(f'The number is num:12,.2f')

  $\rightarrow$ **The number is 12,345.68**

- The above output has field width set to 12.



Formatted Output & F-strings (Cont'd)

- Aligning values within a field:

  Use $<$ for left alignment.

  Use $>$ for right alignment.

  Use $\hat{}$ for center alignment.

- For Example:

  print(f'num:<20.2f')

  print(f'num:>20.2f')

  print(f'num:^20.2f')

```
12345.68
                12345.68

         12345.68
```

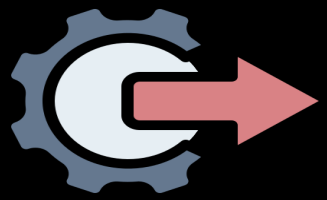
Formatted Output & F-strings (Cont'd)

- The order of designators in a format specifier is important to produce the right output.
- When using multiple designators in a format specifier, write them in this order:

  [alignment][width][,][.precision][type]
- Example:

  print(f'number: ^ 10,.2f')

  $\rightarrow$ 12,345.68



Magic Numbers

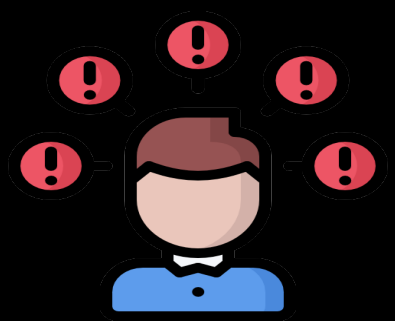
- A magic number is an unexplained numeric value that appears in a program's code.
- Example:

  amount = balance $*$ 0.069
- What is the value 0.069?

  An interest rate? A fee percentage? Only the person who wrote the code knows for sure.

The Problem with Magic Numbers

- It can be difficult to determine the purpose of the number.
- If the **magic number** is used in multiple places in the program, a lot of effort is needed to change the number in each location, if required.
- There is a risk of making a mistake when the magic number is typed in the program's code.
- **For example,** suppose you intend to type 0.069, but you accidentally type .0069. This mistake will cause mathematical errors that can be difficult to find.



Named Constants

- You should use named constants instead of magic numbers.
- A named constant is a name that represents a value that does not change during the program's execution.
- Example:

  INTEREST_RATE = 0.069
- This creates a named constant named **INTEREST_RATE**, assigned the value 0.069.

  It can be used instead of the magic number:

  amount = balance * INTEREST_RATE

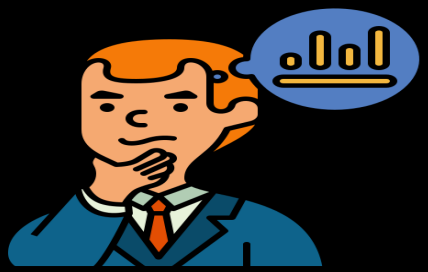
Advantages of Using Named Constants

- Named constants make code self-explanatory (self-documenting).
- Named constants make code easier to maintain (change the value assigned to the constant, and the new value takes effect everywhere the constant is used).
- Named constants help prevent typographical errors that are common when using magic numbers.
- You should use named constants instead of magic numbers.

Lesson Summary

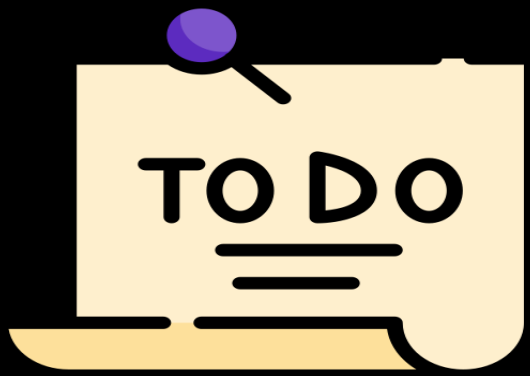
This chapter covered:

- String Concatenation.
- More About The print Function.
- Displaying Formatted Output.
- Named Constants.



Things to do

- Complete Acitivty for Week-3.
- Do more progress in Assignment-1.
- Read Textbook Chapter-2.



Questions?

Please ask your Questions to clarify!